****

Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інтегрованих інформаційних систем

Лабораторна робота №4  
**Мережеве програмування у середовищі UNIX**

Тема: «TCP клієнт-сервер з мультиплексуванням введення-виведення*»*

|  |  |
| --- | --- |
| Виконав:  Студент групи ІА-12  Оверчук Дмитро Максимович | Перевірив:  Сімоненко А.В. |

Київ 2025

## Завдання на роботу

Розробити однопотоковий сервер, який виконує наступне:

1. Сервер підтримує аргументи командного рядка, визначені в лабораторній роботі No3. Та- кож сервер підтримує аргумент командного рядка, який визначає максимальну кількість клієнтів, з якими сервер може одночасно працювати. Сервер не приймає нові TCP з’єдна- ння після досягнення цього значення.
2. Сервер працює з клієнтами відповідно до користувальницького протоколу, визначеного в лабораторній роботі No3.
3. Сервер дозволяє одночасно працювати з кількома клієнтами за допомогою мультипле- ксування введення-виведення. Сервер послуговується системними викликами select() або poll() для мультиплексування введення-виведення.
4. Кількість даних, які сервер зчитує або відправляє одному клієнту під час виконання введення-виведення з ним, треба обмежити. Ця кількість задається в коді сервера кон-стантою, яка може мати значення 1 байт та більше. Тобто, якщо сервер отримав інформа- цію від ядра про можливість виконати введення або виведення для якогось дескриптора файлу сокета, тоді серверу дозволено відправити або отримати даних розміром не більше вказаної константи. Це обмеження дає змогу майже порівну розподіляти час роботи сервера для кожного клієнта, який потребує комунікації. Також невеликі значення цієї константи дозволяють імітувати проблеми з мережею та частково імітувати різну поведінку клієнтів.

Сервер не має завершувати своє виконання у випадку виникнення несистемної помилки.

Рекомендації для сервера такі самі, які були дані в лабораторній роботі No3.

## Код програми

[client](https://github.com/dymytryke/Netw.UNIX/blob/lab4/4/src/client.c)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdint.h>

#include <errno.h>

#include <fcntl.h>

#include "protocol.h"

**//** Function to **print** error messages

void print\_error**(**const char **\***message**)** **{**

perror**(**message**);**

**exit(**EXIT\_FAILURE**);**

**}**

**//** Function to **print** error code description

void print\_error\_code**(int** code**)** **{**

printf**(**"Error: "**);**

switch **(**code**)** **{**

case ERR\_PROTOCOL\_MISMATCH**:**

printf**(**"Protocol version mismatch\n"**);**

**break;**

case ERR\_INVALID\_FILENAME**:**

printf**(**"Invalid filename\n"**);**

**break;**

case ERR\_FILE\_NOT\_FOUND**:**

printf**(**"File not found\n"**);**

**break;**

case ERR\_FILE\_ACCESS**:**

printf**(**"Access to file denied\n"**);**

**break;**

case ERR\_INTERNAL**:**

printf**(**"Internal server error\n"**);**

**break;**

default**:**

printf**(**"Unknown error (code %d)\n"**,** code**);**

**}**

**}**

**//** Main function

**int** main**(int** argc**,** char **\***argv**[])** **{**

**if** **(**argc **!=** 5**)** **{**

fprintf**(**stderr**,** "Usage: %s <server\_address> <server\_port> <filename> <max\_file\_size>\n"**,** argv**[**0**]);**

**exit(**EXIT\_FAILURE**);**

**}**

**//** Parse command line arguments

const char **\***server\_address **=** argv**[**1**];**

**int** server\_port **=** atoi**(**argv**[**2**]);**

const char **\***filename **=** argv**[**3**];**

uint64\_t max\_file\_size **=** strtoull**(**argv**[**4**],** NULL**,** 10**);**

**//** Validate filename length

size\_t filename\_len **=** strlen**(**filename**);**

**if** **(**filename\_len **==** 0 **||** filename\_len **>** MAX\_FILENAME\_LENGTH**)** **{**

fprintf**(**stderr**,** "Error: Filename must be between 1 and %d characters\n"**,** MAX\_FILENAME\_LENGTH**);**

**exit(**EXIT\_FAILURE**);**

**}**

**//** Create socket

**int** sock **=** socket**(**AF\_INET**,** SOCK\_STREAM**,** 0**);**

**if** **(**sock **<** 0**)** **{**

print\_error**(**"Socket creation failed"**);**

**}**

**//** Connect to server

struct sockaddr\_in server\_addr**;**

memset**(&**server\_addr**,** 0**,** sizeof**(**server\_addr**));**

server\_addr**.***sin\_family* **=** AF\_INET**;**

server\_addr**.***sin\_port* **=** htons**(**server\_port**);**

**if** **(**inet\_pton**(**AF\_INET**,** server\_address**,** **&**server\_addr**.***sin\_addr***)** **<=** 0**)** **{**

fprintf**(**stderr**,** "Invalid address/ Address not supported\n"**);**

close**(**sock**);**

**exit(**EXIT\_FAILURE**);**

**}**

printf**(**"Connecting to server at %s:%d\n"**,** server\_address**,** server\_port**);**

**if** **(**connect**(**sock**,** **(**struct sockaddr **\*)&**server\_addr**,** sizeof**(**server\_addr**))** **<** 0**)** **{**

print\_error**(**"Connection failed"**);**

**}**

printf**(**"Connected to server\n"**);**

**//** Prepare file request

FileRequestHeader request**;**

memset**(&**request**,** 0**,** sizeof**(**request**));**

request**.***protocol\_version* **=** PROTOCOL\_VERSION**;**

request**.***message\_type* **=** MSG\_FILE\_REQUEST**;**

request**.***filename\_length* **=** filename\_len**;**

strncpy**(**request**.***filename***,** filename**,** MAX\_FILENAME\_LENGTH**);**

printf**(**"Sending file request for: %s\n"**,** filename**);**

**//** Send file request

**if** **(**send**(**sock**,** **&**request**,** sizeof**(**request**),** 0**)** **!=** sizeof**(**request**))** **{**

print\_error**(**"Failed to send file request"**);**

**}**

**//** Receive server response

FileResponseHeader response**;**

ssize\_t bytes\_received **=** recv**(**sock**,** **&**response**,** sizeof**(**response**),** 0**);**

**if** **(**bytes\_received **!=** sizeof**(**response**))** **{**

print\_error**(**"Failed to receive response header"**);**

**}**

printf**(**"Received response: protocol version %d, message type %d, status %d, file size %lu\n"**,**

response**.***protocol\_version***,** response**.***message\_type***,** response**.***status***,** response**.***file\_size***);**

**//** Check protocol version

**if** **(**response**.***protocol\_version* **!=** PROTOCOL\_VERSION**)** **{**

fprintf**(**stderr**,** "Error: Protocol version mismatch\n"**);**

close**(**sock**);**

**exit(**EXIT\_FAILURE**);**

**}**

**//** Check **for** error

**if** **(**response**.***status* **!=** 0**)** **{**

print\_error\_code**(**response**.***status***);**

close**(**sock**);**

**exit(**EXIT\_FAILURE**);**

**}**

**//** Check file size against maximum

ClientResponseHeader client\_response**;**

**if** **(**response**.***file\_size* **>** max\_file\_size**)** **{**

printf**(**"File size (%lu bytes) exceeds maximum allowed size (%lu bytes)\n"**,**

response**.***file\_size***,** max\_file\_size**);**

client\_response**.***message\_type* **=** MSG\_REFUSE\_TO\_RECEIVE**;**

send**(**sock**,** **&**client\_response**,** sizeof**(**client\_response**),** 0**);**

close**(**sock**);**

**exit(**EXIT\_FAILURE**);**

**}**

**//** Send ready to receive

client\_response**.***message\_type* **=** MSG\_READY\_TO\_RECEIVE**;**

printf**(**"Sending ready to receive message\n"**);**

**if** **(**send**(**sock**,** **&**client\_response**,** sizeof**(**client\_response**),** 0**)** **!=** sizeof**(**client\_response**))** **{**

print\_error**(**"Failed to send ready message"**);**

**}**

**//** Create output file

**int** file\_fd **=** **open(**filename**,** O\_WRONLY **|** O\_CREAT **|** O\_TRUNC**,** 0644**);**

**if** **(**file\_fd **<** 0**)** **{**

print\_error**(**"Failed to create output file"**);**

**}**

**//** Receive file content

printf**(**"Receiving file content (%lu bytes)...\n"**,** response**.***file\_size***);**

uint64\_t total\_received **=** 0**;**

char buffer**[**4096**];**

**while** **(**total\_received **<** response**.***file\_size***)** **{**

bytes\_received **=** recv**(**sock**,** buffer**,** sizeof**(**buffer**),** 0**);**

**if** **(**bytes\_received **<=** 0**)** **{**

**if** **(**bytes\_received **==** 0**)** **{**

printf**(**"Connection closed by server\n"**);**

**break;**

**}** **else** **{**

print\_error**(**"Error receiving file content"**);**

**}**

**}**

**if** **(**write**(**file\_fd**,** buffer**,** bytes\_received**)** **!=** bytes\_received**)** **{**

print\_error**(**"Failed to write to output file"**);**

**}**

total\_received **+=** bytes\_received**;**

printf**(**"\rReceived: %lu/%lu bytes (%.1f%%)"**,**

total\_received**,** response**.***file\_size***,**

**(float)**total\_received **/** response**.***file\_size* **\*** 100**);**

fflush**(**stdout**);**

**}**

printf**(**"\nFile transfer complete\n"**);**

**//** Close file **and** socket

close**(**file\_fd**);**

close**(**sock**);**

**return** 0**;**

**}**

[server](https://github.com/dymytryke/Netw.UNIX/blob/lab4/4/src/multiplexing_server.c)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/socket.h>

#include <sys/poll.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <errno.h>

#include <sys/stat.h>

#include <dirent.h>

#include <limits.h>

#include <time.h>

#include "protocol.h"

#include "utils.h"

**//** Maximum data chunk to read**/**write **in** a single operation

#define MAX\_IO\_CHUNK 512

**//** Client connection states

typedef enum **{**

STATE\_READ\_REQUEST**,** **//** Reading file request **from** client

STATE\_SEND\_RESPONSE**,** **//** Sending file response to client

STATE\_WAIT\_DECISION**,** **//** Waiting **for** client's decision to receive file

STATE\_SEND\_FILE**,** **//** Sending file content to client

STATE\_DONE **//** Client handled**,** connection can be closed

**}** ClientState**;**

**//** Client context structure

typedef struct **{**

**int** socket**;** **//** Client socket descriptor

ClientState state**;** **//** Current state

time\_t last\_activity**;** **//** Time of last activity

**//** Request handling

FileRequestHeader request**;** **//** Client request

size\_t request\_bytes\_read**;** **//** How much of the request has been read

**//** Response handling

FileResponseHeader response**;** **//** Server response

size\_t response\_bytes\_sent**;** **//** How much of the response has been sent

**//** Client decision handling

ClientResponseHeader client\_response**;** **//** Client's decision response

size\_t decision\_bytes\_read**;** **//** How much of the decision has been read

**//** File transfer

**int** file\_fd**;** **//** File descriptor

uint64\_t file\_size**;** **//** Size of the file

uint64\_t file\_bytes\_sent**;** **//** How much of the file has been sent

char filepath**[**PATH\_MAX**];** **//** Full path to the file

char buffer**[**MAX\_IO\_CHUNK**];** **//** Buffer **for** I**/**O operations

**}** ClientContext**;**

**//** Global variables

char server\_directory**[**PATH\_MAX**];** **//** Directory to serve files **from**

**int** max\_clients **=** 0**;** **//** Maximum number of clients

**int** active\_clients **=** 0**;** **//** Number of active client connections

**//** Function prototypes

void setup\_server\_socket**(int** **\***server\_sock**,** const char **\***server\_address**,** **int** server\_port**);**

void set\_nonblocking**(int** socket\_fd**);**

**int** accept\_new\_client**(int** server\_sock**,** struct pollfd **\***fds**,** ClientContext **\***contexts**,** **int** **\***nfds**);**

void handle\_client\_io**(**struct pollfd **\***fd**,** ClientContext **\***context**);**

void cleanup\_client**(**struct pollfd **\***fds**,** ClientContext **\***contexts**,** **int** index**,** **int** **\***nfds**);**

void process\_read\_request**(**struct pollfd **\***fd**,** ClientContext **\***context**);**

void process\_send\_response**(**struct pollfd **\***fd**,** ClientContext **\***context**);**

void process\_wait\_decision**(**struct pollfd **\***fd**,** ClientContext **\***context**);**

void process\_send\_file**(**struct pollfd **\***fd**,** ClientContext **\***context**);**

**//** Main function

**int** main**(int** argc**,** char **\***argv**[])** **{**

**if** **(**argc **!=** 5**)** **{**

fprintf**(**stderr**,** "Usage: %s <server\_address> <server\_port> <directory> <max\_clients>\n"**,** argv**[**0**]);**

**exit(**EXIT\_FAILURE**);**

**}**

**//** Parse command line arguments

const char **\***server\_address **=** argv**[**1**];**

**int** server\_port **=** atoi**(**argv**[**2**]);**

strncpy**(**server\_directory**,** argv**[**3**],** sizeof**(**server\_directory**)** **-** 1**);**

max\_clients **=** atoi**(**argv**[**4**]);**

**if** **(**max\_clients **<=** 0**)** **{**

fprintf**(**stderr**,** "Error: max\_clients must be a positive integer\n"**);**

**exit(**EXIT\_FAILURE**);**

**}**

**//** Check **if** directory exists

DIR **\*dir** **=** opendir**(**server\_directory**);**

**if** **(!dir)** **{**

fprintf**(**stderr**,** "Error: Directory '%s' does not exist or is not accessible\n"**,** server\_directory**);**

**exit(**EXIT\_FAILURE**);**

**}**

closedir**(dir);**

**//** Create **and** setup server socket

**int** server\_sock**;**

setup\_server\_socket**(&**server\_sock**,** server\_address**,** server\_port**);**

**//** Set server socket to non**-**blocking mode

set\_nonblocking**(**server\_sock**);**

**//** Allocate memory **for** poll structures **and** client contexts

**//** **+**1 **for** server socket

struct pollfd **\***fds **=** malloc**((**max\_clients **+** 1**)** **\*** sizeof**(**struct pollfd**));**

ClientContext **\***contexts **=** malloc**((**max\_clients **+** 1**)** **\*** sizeof**(**ClientContext**));**

**if** **(!**fds **||** **!**contexts**)** **{**

perror**(**"Failed to allocate memory"**);**

**exit(**EXIT\_FAILURE**);**

**}**

**//** Initialize server socket **in** poll structure

fds**[**0**].***fd* **=** server\_sock**;**

fds**[**0**].***events* **=** POLLIN**;**

**int** nfds **=** 1**;**

printf**(**"Multiplexing server started at %s:%d, serving files from '%s', max clients: %d\n"**,**

server\_address**,** server\_port**,** server\_directory**,** max\_clients**);**

**//** Main server loop

**while** **(**1**)** **{**

**//** Wait **for** events on sockets

**int** poll\_result **=** poll**(**fds**,** nfds**,** **-**1**);**

**if** **(**poll\_result **<** 0**)** **{**

**if** **(**errno **==** EINTR**)** **{**

**continue;** **//** Interrupted system call**,** **try** again

**}**

perror**(**"Poll failed"**);**

**break;**

**}**

**//** Check **for** events on **all** file descriptors

**for** **(int** i **=** 0**;** i **<** nfds**;** i**++)** **{**

**if** **(**fds**[**i**].***revents* **==** 0**)** **{**

**continue;** **//** No events on this descriptor

**}**

**//** Check **for** error conditions

**if** **(**fds**[**i**].***revents* **&** **(**POLLERR **|** POLLHUP **|** POLLNVAL**))** **{**

**if** **(**i **==** 0**)** **{**

**//** Error on server socket**,** critical error

fprintf**(**stderr**,** "Error on server socket, exiting\n"**);**

**exit(**EXIT\_FAILURE**);**

**}** **else** **{**

**//** Error on client socket**,** close connection

printf**(**"Socket error (fd=%d), closing connection\n"**,** fds**[**i**].***fd***);**

cleanup\_client**(**fds**,** contexts**,** i**,** **&**nfds**);**

**continue;**

**}**

**}**

**//** Handle incoming connection on server socket

**if** **(**i **==** 0 **&&** **(**fds**[**i**].***revents* **&** POLLIN**))** **{**

**if** **(**active\_clients **<** max\_clients**)** **{**

**if** **(**accept\_new\_client**(**server\_sock**,** fds**,** contexts**,** **&**nfds**))** **{**

active\_clients**++;**

**}**

**}**

**continue;**

**}**

**//** Handle client I**/**O

**if** **(**fds**[**i**].***revents* **&** **(**POLLIN **|** POLLOUT**))** **{**

handle\_client\_io**(&**fds**[**i**],** **&**contexts**[**i**]);**

**//** Check **if** client **is** done **and** needs to be closed

**if** **(**contexts**[**i**].***state* **==** STATE\_DONE**)** **{**

cleanup\_client**(**fds**,** contexts**,** i**,** **&**nfds**);**

active\_clients**--;**

**}** **else** **{**

**//** Update last activity time

contexts**[**i**].***last\_activity* **=** time**(**NULL**);**

**}**

**}**

**}**

**}**

**//** Cleanup

close**(**server\_sock**);**

free**(**fds**);**

free**(**contexts**);**

**return** 0**;**

**}**

**//** Set up server socket

void setup\_server\_socket**(int** **\***server\_sock**,** const char **\***server\_address**,** **int** server\_port**)** **{**

**\***server\_sock **=** socket**(**AF\_INET**,** SOCK\_STREAM**,** 0**);**

**if** **(\***server\_sock **<** 0**)** **{**

perror**(**"Socket creation failed"**);**

**exit(**EXIT\_FAILURE**);**

**}**

**//** Set socket option to reuse address

**int** opt **=** 1**;**

**if** **(**setsockopt**(\***server\_sock**,** SOL\_SOCKET**,** SO\_REUSEADDR**,** **&**opt**,** sizeof**(**opt**))** **<** 0**)** **{**

perror**(**"Setsockopt failed"**);**

**exit(**EXIT\_FAILURE**);**

**}**

**//** Bind socket to specified address **and** port

struct sockaddr\_in server\_addr**;**

memset**(&**server\_addr**,** 0**,** sizeof**(**server\_addr**));**

server\_addr**.***sin\_family* **=** AF\_INET**;**

server\_addr**.***sin\_port* **=** htons**(**server\_port**);**

**if** **(**inet\_pton**(**AF\_INET**,** server\_address**,** **&**server\_addr**.***sin\_addr***)** **<=** 0**)** **{**

fprintf**(**stderr**,** "Invalid address/ Address not supported\n"**);**

**exit(**EXIT\_FAILURE**);**

**}**

**if** **(**bind**(\***server\_sock**,** **(**struct sockaddr **\*)&**server\_addr**,** sizeof**(**server\_addr**))** **<** 0**)** **{**

perror**(**"Bind failed"**);**

**exit(**EXIT\_FAILURE**);**

**}**

**//** Listen **for** connections

**if** **(**listen**(\***server\_sock**,** max\_clients**)** **<** 0**)** **{**

perror**(**"Listen failed"**);**

**exit(**EXIT\_FAILURE**);**

**}**

**}**

**//** Set socket to non**-**blocking mode

void set\_nonblocking**(int** socket\_fd**)** **{**

**int** flags **=** fcntl**(**socket\_fd**,** F\_GETFL**,** 0**);**

**if** **(**flags **==** **-**1**)** **{**

perror**(**"fcntl F\_GETFL"**);**

**exit(**EXIT\_FAILURE**);**

**}**

**if** **(**fcntl**(**socket\_fd**,** F\_SETFL**,** flags **|** O\_NONBLOCK**)** **==** **-**1**)** **{**

perror**(**"fcntl F\_SETFL O\_NONBLOCK"**);**

**exit(**EXIT\_FAILURE**);**

**}**

**}**

**//** Accept new client connection

**int** accept\_new\_client**(int** server\_sock**,** struct pollfd **\***fds**,** ClientContext **\***contexts**,** **int** **\***nfds**)** **{**

struct sockaddr\_in client\_addr**;**

socklen\_t client\_len **=** sizeof**(**client\_addr**);**

**//** Accept connection

**int** client\_sock **=** accept**(**server\_sock**,** **(**struct sockaddr **\*)&**client\_addr**,** **&**client\_len**);**

**if** **(**client\_sock **<** 0**)** **{**

**if** **(**errno **==** EAGAIN **||** errno **==** EWOULDBLOCK**)** **{**

**return** 0**;** **//** No pending connections

**}**

perror**(**"Accept failed"**);**

**return** 0**;**

**}**

**//** Set client socket to non**-**blocking mode

set\_nonblocking**(**client\_sock**);**

**//** Get client information

char client\_ip**[**INET\_ADDRSTRLEN**];**

inet\_ntop**(**AF\_INET**,** **&**client\_addr**.***sin\_addr***,** client\_ip**,** INET\_ADDRSTRLEN**);**

**int** client\_port **=** ntohs**(**client\_addr**.***sin\_port***);**

printf**(**"Accepted connection from %s:%d (fd=%d)\n"**,** client\_ip**,** client\_port**,** client\_sock**);**

**//** Initialize client context

**int** index **=** **\***nfds**;**

fds**[**index**].***fd* **=** client\_sock**;**

fds**[**index**].***events* **=** POLLIN**;** **//** Initially**,** wait **for** client request

memset**(&**contexts**[**index**],** 0**,** sizeof**(**ClientContext**));**

contexts**[**index**].***socket* **=** client\_sock**;**

contexts**[**index**].***state* **=** STATE\_READ\_REQUEST**;**

contexts**[**index**].***last\_activity* **=** time**(**NULL**);**

contexts**[**index**].***file\_fd* **=** **-**1**;**

**(\***nfds**)++;**

**return** 1**;**

**}**

**//** Handle client I**/**O based on current state

void handle\_client\_io**(**struct pollfd **\***fd**,** ClientContext **\***context**)** **{**

switch **(**context**->**state**)** **{**

case STATE\_READ\_REQUEST**:**

process\_read\_request**(**fd**,** context**);**

**break;**

case STATE\_SEND\_RESPONSE**:**

process\_send\_response**(**fd**,** context**);**

**break;**

case STATE\_WAIT\_DECISION**:**

process\_wait\_decision**(**fd**,** context**);**

**break;**

case STATE\_SEND\_FILE**:**

process\_send\_file**(**fd**,** context**);**

**break;**

case STATE\_DONE**:**

**//** Nothing to do**,** will be cleaned up **in** main loop

**break;**

**}**

**}**

**//** Process reading client request

void process\_read\_request**(**struct pollfd **\***fd**,** ClientContext **\***context**)** **{**

**//** Calculate how many **bytes** we still need to read

size\_t bytes\_to\_read **=** sizeof**(**FileRequestHeader**)** **-** context**->**request\_bytes\_read**;**

**//** Limit by MAX\_IO\_CHUNK

**if** **(**bytes\_to\_read **>** MAX\_IO\_CHUNK**)**

bytes\_to\_read **=** MAX\_IO\_CHUNK**;**

**//** Read **from** socket

char **\***buffer\_ptr **=** **(**char **\*)&**context**->**request **+** context**->**request\_bytes\_read**;**

ssize\_t bytes\_read **=** read**(**context**->**socket**,** buffer\_ptr**,** bytes\_to\_read**);**

**if** **(**bytes\_read **<** 0**)** **{**

**if** **(**errno **==** EAGAIN **||** errno **==** EWOULDBLOCK**)** **{**

**return;** **//** No data available**,** **try** again later

**}**

perror**(**"Read error"**);**

context**->**state **=** STATE\_DONE**;**

**return;**

**}**

**if** **(**bytes\_read **==** 0**)** **{**

printf**(**"Client closed connection during request read\n"**);**

context**->**state **=** STATE\_DONE**;**

**return;**

**}**

context**->**request\_bytes\_read **+=** bytes\_read**;**

**//** If we've read the entire request

**if** **(**context**->**request\_bytes\_read **==** sizeof**(**FileRequestHeader**))** **{**

**//** Prepare response

memset**(&**context**->**response**,** 0**,** sizeof**(**FileResponseHeader**));**

context**->**response**.***protocol\_version* **=** PROTOCOL\_VERSION**;**

context**->**response**.***message\_type* **=** MSG\_FILE\_RESPONSE**;**

**//** Check protocol version

**if** **(**context**->**request**.***protocol\_version* **!=** PROTOCOL\_VERSION**)** **{**

printf**(**"Error: Protocol version mismatch (client: %d, server: %d)\n"**,**

context**->**request**.***protocol\_version***,** PROTOCOL\_VERSION**);**

context**->**response**.***status* **=** ERR\_PROTOCOL\_MISMATCH**;**

context**->**state **=** STATE\_SEND\_RESPONSE**;**

fd**->**events **=** POLLOUT**;**

**return;**

**}**

**//** Null**-**terminate filename to be safe

context**->**request**.***filename***[**context**->**request**.***filename\_length***]** **=** '\0'**;**

printf**(**"Received file request from fd=%d for: %s\n"**,** context**->**socket**,** context**->**request**.***filename***);**

**//** Validate filename

**if** **(!**validate\_filename**(**context**->**request**.***filename***))** **{**

printf**(**"Error: Invalid filename: %s\n"**,** context**->**request**.***filename***);**

context**->**response**.***status* **=** ERR\_INVALID\_FILENAME**;**

context**->**state **=** STATE\_SEND\_RESPONSE**;**

fd**->**events **=** POLLOUT**;**

**return;**

**}**

**//** Build full path

snprintf**(**context**->**filepath**,** sizeof**(**context**->**filepath**),** "%s/%s"**,**

server\_directory**,** context**->**request**.***filename***);**

**//** Check **if** the file exists **and** **is** a regular file

struct stat file\_stat**;**

**if** **(**stat**(**context**->**filepath**,** **&**file\_stat**)** **!=** 0**)** **{**

printf**(**"Error: File not found: %s\n"**,** context**->**filepath**);**

context**->**response**.***status* **=** ERR\_FILE\_NOT\_FOUND**;**

context**->**state **=** STATE\_SEND\_RESPONSE**;**

fd**->**events **=** POLLOUT**;**

**return;**

**}**

**if** **(!**S\_ISREG**(**file\_stat**.***st\_mode***))** **{**

printf**(**"Error: Not a regular file: %s\n"**,** context**->**filepath**);**

context**->**response**.***status* **=** ERR\_FILE\_NOT\_FOUND**;**

context**->**state **=** STATE\_SEND\_RESPONSE**;**

fd**->**events **=** POLLOUT**;**

**return;**

**}**

**//** Get file size

context**->**file\_size **=** file\_stat**.***st\_size***;**

**//** Prepare success response

context**->**response**.***status* **=** 0**;** **//** Success

context**->**response**.***file\_size* **=** context**->**file\_size**;**

printf**(**"Sending file response to fd=%d: status=%d, file\_size=%lu\n"**,**

context**->**socket**,** context**->**response**.***status***,** context**->**response**.***file\_size***);**

**//** Change state **and** update poll events

context**->**state **=** STATE\_SEND\_RESPONSE**;**

fd**->**events **=** POLLOUT**;**

**}**

**}**

**//** Process sending response to client

void process\_send\_response**(**struct pollfd **\***fd**,** ClientContext **\***context**)** **{**

**//** Calculate how many **bytes** we still need to send

size\_t bytes\_to\_send **=** sizeof**(**FileResponseHeader**)** **-** context**->**response\_bytes\_sent**;**

**//** Limit by MAX\_IO\_CHUNK

**if** **(**bytes\_to\_send **>** MAX\_IO\_CHUNK**)**

bytes\_to\_send **=** MAX\_IO\_CHUNK**;**

**//** Write to socket

char **\***buffer\_ptr **=** **(**char **\*)&**context**->**response **+** context**->**response\_bytes\_sent**;**

ssize\_t bytes\_sent **=** write**(**context**->**socket**,** buffer\_ptr**,** bytes\_to\_send**);**

**if** **(**bytes\_sent **<** 0**)** **{**

**if** **(**errno **==** EAGAIN **||** errno **==** EWOULDBLOCK**)** **{**

**return;** **//** Buffer full**,** **try** again later

**}**

perror**(**"Write error"**);**

context**->**state **=** STATE\_DONE**;**

**return;**

**}**

context**->**response\_bytes\_sent **+=** bytes\_sent**;**

**//** If we've sent the entire response

**if** **(**context**->**response\_bytes\_sent **==** sizeof**(**FileResponseHeader**))** **{**

**//** If response indicated an error**,** we're done

**if** **(**context**->**response**.***status* **!=** 0**)** **{**

context**->**state **=** STATE\_DONE**;**

**return;**

**}**

**//** Wait **for** client's decision

context**->**state **=** STATE\_WAIT\_DECISION**;**

fd**->**events **=** POLLIN**;**

**}**

**}**

**//** Process waiting **for** client decision

void process\_wait\_decision**(**struct pollfd **\***fd**,** ClientContext **\***context**)** **{**

**//** Calculate how many **bytes** we still need to read

size\_t bytes\_to\_read **=** sizeof**(**ClientResponseHeader**)** **-** context**->**decision\_bytes\_read**;**

**//** Limit by MAX\_IO\_CHUNK

**if** **(**bytes\_to\_read **>** MAX\_IO\_CHUNK**)**

bytes\_to\_read **=** MAX\_IO\_CHUNK**;**

**//** Read **from** socket

char **\***buffer\_ptr **=** **(**char **\*)&**context**->**client\_response **+** context**->**decision\_bytes\_read**;**

ssize\_t bytes\_read **=** read**(**context**->**socket**,** buffer\_ptr**,** bytes\_to\_read**);**

**if** **(**bytes\_read **<** 0**)** **{**

**if** **(**errno **==** EAGAIN **||** errno **==** EWOULDBLOCK**)** **{**

**return;** **//** No data available**,** **try** again later

**}**

perror**(**"Read error"**);**

context**->**state **=** STATE\_DONE**;**

**return;**

**}**

**if** **(**bytes\_read **==** 0**)** **{**

printf**(**"Client closed connection during decision read\n"**);**

context**->**state **=** STATE\_DONE**;**

**return;**

**}**

context**->**decision\_bytes\_read **+=** bytes\_read**;**

**//** If we've read the entire decision

**if** **(**context**->**decision\_bytes\_read **==** sizeof**(**ClientResponseHeader**))** **{**

**if** **(**context**->**client\_response**.***message\_type* **==** MSG\_REFUSE\_TO\_RECEIVE**)** **{**

printf**(**"Client fd=%d refused to receive the file (too large)\n"**,** context**->**socket**);**

context**->**state **=** STATE\_DONE**;**

**return;**

**}**

**if** **(**context**->**client\_response**.***message\_type* **!=** MSG\_READY\_TO\_RECEIVE**)** **{**

printf**(**"Error: Unexpected client response message type: %d\n"**,**

context**->**client\_response**.***message\_type***);**

context**->**state **=** STATE\_DONE**;**

**return;**

**}**

printf**(**"Client fd=%d is ready to receive the file\n"**,** context**->**socket**);**

**//** Open the file

context**->**file\_fd **=** **open(**context**->**filepath**,** O\_RDONLY**);**

**if** **(**context**->**file\_fd **<** 0**)** **{**

perror**(**"Failed to open file"**);**

context**->**state **=** STATE\_DONE**;**

**return;**

**}**

**//** Change state to send file **and** update poll events

context**->**state **=** STATE\_SEND\_FILE**;**

context**->**file\_bytes\_sent **=** 0**;**

fd**->**events **=** POLLOUT**;**

**}**

**}**

**//** Process sending file content

void process\_send\_file**(**struct pollfd **\***fd**,** ClientContext **\***context**)** **{**

**//** If we've sent the entire file

**if** **(**context**->**file\_bytes\_sent **>=** context**->**file\_size**)** **{**

printf**(**"\nFile transfer complete for client fd=%d\n"**,** context**->**socket**);**

**//** Close file descriptor **and** mark **as** done

**if** **(**context**->**file\_fd **>=** 0**)** **{**

close**(**context**->**file\_fd**);**

context**->**file\_fd **=** **-**1**;**

**}**

context**->**state **=** STATE\_DONE**;**

**return;**

**}**

**//** Read **from** file into buffer **(**limited by MAX\_IO\_CHUNK**)**

size\_t bytes\_to\_read **=** MAX\_IO\_CHUNK**;**

**if** **(**context**->**file\_size **-** context**->**file\_bytes\_sent **<** bytes\_to\_read**)** **{**

bytes\_to\_read **=** context**->**file\_size **-** context**->**file\_bytes\_sent**;**

**}**

ssize\_t bytes\_read **=** read**(**context**->**file\_fd**,** context**->**buffer**,** bytes\_to\_read**);**

**if** **(**bytes\_read **<=** 0**)** **{**

**if** **(**bytes\_read **<** 0**)** **{**

perror**(**"File read error"**);**

**}**

**//** End of file **or** error

context**->**state **=** STATE\_DONE**;**

close**(**context**->**file\_fd**);**

context**->**file\_fd **=** **-**1**;**

**return;**

**}**

**//** Send buffer to socket

ssize\_t bytes\_sent **=** write**(**context**->**socket**,** context**->**buffer**,** bytes\_read**);**

**if** **(**bytes\_sent **<** 0**)** **{**

**if** **(**errno **==** EAGAIN **||** errno **==** EWOULDBLOCK**)** **{**

**//** Rewind file pointer **as** we couldn't send data

**if** **(**lseek**(**context**->**file\_fd**,** **-**bytes\_read**,** SEEK\_CUR**)** **<** 0**)** **{**

perror**(**"lseek error"**);**

context**->**state **=** STATE\_DONE**;**

close**(**context**->**file\_fd**);**

context**->**file\_fd **=** **-**1**;**

**}**

**return;** **//** Buffer full**,** **try** again later

**}**

perror**(**"Write error"**);**

context**->**state **=** STATE\_DONE**;**

close**(**context**->**file\_fd**);**

context**->**file\_fd **=** **-**1**;**

**return;**

**}**

**if** **(**bytes\_sent **<** bytes\_read**)** **{**

**//** We couldn't send all bytes, rewind file pointer

off\_t rewind\_offset **=** bytes\_read **-** bytes\_sent**;**

**if** **(**lseek**(**context**->**file\_fd**,** **-**rewind\_offset**,** SEEK\_CUR**)** **<** 0**)** **{**

perror**(**"lseek error"**);**

context**->**state **=** STATE\_DONE**;**

close**(**context**->**file\_fd**);**

context**->**file\_fd **=** **-**1**;**

**return;**

**}**

**}**

**//** Update **bytes** sent

context**->**file\_bytes\_sent **+=** bytes\_sent**;**

**//** Print progress **(**only **for** significant changes to avoid console flooding**)**

static **int** last\_percent**[**FD\_SETSIZE**]** **=** **{**0**};**

**int** current\_percent **=** **(int)((**double**)**context**->**file\_bytes\_sent **/** context**->**file\_size **\*** 100**);**

**if** **(**current\_percent **>** last\_percent**[**context**->**socket**]** **||**

context**->**file\_bytes\_sent **>=** context**->**file\_size**)** **{**

printf**(**"\rClient fd=%d: Sent %lu/%lu bytes (%d%%)"**,**

context**->**socket**,** context**->**file\_bytes\_sent**,** context**->**file\_size**,** current\_percent**);**

fflush**(**stdout**);**

last\_percent**[**context**->**socket**]** **=** current\_percent**;**

**}**

**}**

**//** Clean up client resources **and** remove **from** poll array

void cleanup\_client**(**struct pollfd **\***fds**,** ClientContext **\***contexts**,** **int** index**,** **int** **\***nfds**)** **{**

**//** Close file descriptor **if** **open**

**if** **(**contexts**[**index**].***file\_fd* **>=** 0**)** **{**

close**(**contexts**[**index**].***file\_fd***);**

**}**

**//** Close socket

close**(**fds**[**index**].***fd***);**

printf**(**"Closed connection for client fd=%d\n"**,** fds**[**index**].***fd***);**

**//** Remove **from** poll array by shifting remaining descriptors

**(\***nfds**)--;**

**for** **(int** i **=** index**;** i **<** **\***nfds**;** i**++)** **{**

fds**[**i**]** **=** fds**[**i **+** 1**];**

contexts**[**i**]** **=** contexts**[**i **+** 1**];**

**}**

**}**

[protocol](https://github.com/dymytryke/Netw.UNIX/blob/lab4/4/src/protocol.h)

#ifndef PROTOCOL\_H

#define PROTOCOL\_H

#include <stdint.h>

**//** Версія протоколу

#define PROTOCOL\_VERSION 1

**//** Максимальна довжина імені файлу **(**без нульового символу**)**

#define MAX\_FILENAME\_LENGTH 255

**//** Коди помилок

#define ERR\_PROTOCOL\_MISMATCH 1

#define ERR\_INVALID\_FILENAME 2

#define ERR\_FILE\_NOT\_FOUND 3

#define ERR\_FILE\_ACCESS 4

#define ERR\_INTERNAL 5

**//** Типи повідомлень

#define MSG\_FILE\_REQUEST 1

#define MSG\_FILE\_RESPONSE 2

#define MSG\_READY\_TO\_RECEIVE 3

#define MSG\_REFUSE\_TO\_RECEIVE 4

**//** Розмір блоку при передачі файлу **(**64КБ**)**

#define DEFAULT\_CHUNK\_SIZE (64 \* 1024)

**//** Заголовок запиту від клієнта до сервера

typedef struct **{**

uint8\_t protocol\_version**;**

uint8\_t message\_type**;**

uint16\_t filename\_length**;** **//** Довжина імені файлу

char filename**[**MAX\_FILENAME\_LENGTH **+** 1**];** **//** **+**1 для термінатора '\0'

**}** FileRequestHeader**;**

**//** Заголовок відповіді від сервера до клієнта

typedef struct **{**

uint8\_t protocol\_version**;**

uint8\_t message\_type**;**

uint8\_t status**;** **//** 0 **=** успіх**,** інакше код помилки

uint64\_t file\_size**;** **//** Розмір файлу у байтах

**}** FileResponseHeader**;**

**//** Заголовок відповіді клієнта серверу

typedef struct **{**

uint8\_t message\_type**;** **//** MSG\_READY\_TO\_RECEIVE або MSG\_REFUSE\_TO\_RECEIVE

**}** ClientResponseHeader**;**

#endif // PROTOCOL\_H

[utils](https://github.com/dymytryke/Netw.UNIX/blob/lab4/4/src/utils.h)

#ifndef UTILS\_H

#define UTILS\_H

#include <string.h>

**//** Function to validate a filename **(**no path separators **or** invalid characters**)**

static inline **int** validate\_filename**(**const char **\***filename**)** **{**

**//** Check **for** null **or** empty filename

**if** **(!**filename **||** **!\***filename**)** **{**

**return** 0**;**

**}**

**//** Check **for** path separators

**if** **(**strchr**(**filename**,** '/'**)** **!=** NULL**)** **{**

**return** 0**;**

**}**

**//** Check **for** "." **or** ".." which could be used to navigate directory structure

**if** **(**strcmp**(**filename**,** "."**)** **==** 0 **||** strcmp**(**filename**,** ".."**)** **==** 0**)** **{**

**return** 0**;**

**}**

**//** Basic check **for** valid filename characters

**for** **(**const char **\***p **=** filename**;** **\***p**;** p**++)** **{**

**if** **(\***p **<=** 31 **||** **\***p **==** 127**)** **{** **//** Control characters

**return** 0**;**

**}**

**//** This **is** a simplified check **-** actual filesystem restrictions may vary

**if** **(**strchr**(**"<>:\"|?\*\\"**,** **\***p**)** **!=** NULL**)** **{**

**return** 0**;**

**}**

**}**

**return** 1**;**

**}**

#endif // UTILS\_H

## Опис програми

Програма представляє собою однопотоковий мультиплексований сервер для передачі файлів. Розглянемо детальніше особливості його реалізації:

### Архітектурні рішення

Сервер використовує **мультиплексування вводу-виводу** замість створення окремих потоків/процесів для кожного клієнта. Це дозволяє одному потоку виконання обробляти багато з’єднань одночасно. У реалізації використано системний виклик poll(), який відстежує стан усіх сокетів.

### Основні компоненти сервера:

1. **Структура ClientContext** - зберігає стан взаємодії з кожним клієнтом:
   * поточний стан обміну (читання запиту, відправка відповіді тощо)
   * буфер для введення-виведення
   * стан передачі файлу і метадані
   * інформація про активність клієнта
2. **Кінцевий автомат станів клієнта** реалізовано через enum ClientState:
   * STATE\_READ\_REQUEST - читання запиту на файл
   * STATE\_SEND\_RESPONSE - відправка відповіді з метаінформацією
   * STATE\_WAIT\_DECISION - очікування рішення клієнта
   * STATE\_SEND\_FILE - передача файлу
   * STATE\_DONE - завершення обробки

### Особливості реалізації

#### 1. Обмеження на розмір операцій вводу-виводу

Ключовою особливістю є константа MAX\_IO\_CHUNK (512 байт), яка обмежує розмір даних для операцій читання/запису за одну ітерацію. Це забезпечує:

* **Справедливий розподіл ресурсів** - сервер не “зависає” на обробці великого файлу для одного клієнта
* **Імітацію мережевих умов** - дозволяє тестувати роботу з повільними з’єднаннями
* **Рівномірне обслуговування клієнтів** - кожен клієнт отримує обмежену порцію уваги сервера

#### 2. Неблокуючий ввід-вивід

Усі сокети налаштовуються в неблокуючий режим через функцію set\_nonblocking(). Це дозволяє функціям читання/запису повертати керування негайно, навіть якщо дані недоступні.

#### 3. Обмеження кількості клієнтів

Сервер підтримує параметр max\_clients, який обмежує кількість одночасних підключень. При досягненні ліміту нові з’єднання не приймаються.

#### 4. Системний виклик poll()

Центральна частина мультиплексування - використання poll() для моніторингу кількох файлових дескрипторів. Сервер відстежує готовність дескрипторів до операцій читання або запису, а також помилки.

#### 5. Поетапна обробка протоколу

Обробка протоколу розбита на окремі функції згідно зі станами клієнта: - process\_read\_request() - обробка вхідних запитів на файл - process\_send\_response() - відправка метаінформації про файл - process\_wait\_decision() - очікування рішення клієнта - process\_send\_file() - відправка вмісту файлу частинами

### Алгоритм роботи сервера

1. **Ініціалізація**:
   * Розбір параметрів командного рядка
   * Створення сокету та прив’язка до адреси і порту
   * Встановлення сокету в режим прослуховування
   * Налаштування неблокуючого режиму
2. **Основний цикл**:
   * Очікування подій на всіх дескрипторах через poll()
   * Обробка нових підключень (якщо не досягнуто ліміту)
   * Обробка готових для вводу-виводу дескрипторів
   * Обробка помилок і закриття з’єднань
   * Оновлення структур даних і звільнення ресурсів для завершених з’єднань
3. **Обробка клієнтів**:
   * Прийом запиту на файл частинами (до MAX\_IO\_CHUNK за раз)
   * Перевірка існування файлу і підготовка відповіді
   * Відправка відповіді з метаінформацією (частинами)
   * Очікування підтвердження клієнта
   * Передача вмісту файлу невеликими блоками

Як видно з прикладів використання, сервер здатен одночасно обслуговувати декілька клієнтів, обмежуючи їх максимальну кількість і розподіляючи ресурси процесора та мережі між ними. Це дозволяє ефективно передавати файли навіть при роботі з декількома клієнтами одночасно.

## Приклади використання програми

### Запуск сервера

dymytryke@dymytryke:/mnt/c/Users/dymyt/OneDrive/Документы/save/Учеба/4\_курс/Netw.UNIX/4/bin$ ./multiplexing\_server 127.0.0.1 12345 ../test\_files/ 3  
Multiplexing server started at 127.0.0.1:12345, serving files from '../test\_files/', max clients: 3

### Робота з одним клієнтом

**Клієнт**

dymytryke@dymytryke:/mnt/c/Users/dymyt/OneDrive/Документы/save/Учеба/4\_курс/Netw.UNIX/4/bin$ ./client 127.0.0.1 12345 baseball-cap-collection.zip 1200000000  
Connecting to server at 127.0.0.1:12345  
Connected to server  
Sending file request for: baseball-cap-collection.zip  
Received response: protocol version 1, message type 2, status 0, file size 13142141  
Sending ready to receive message  
Receiving file content (13142141 bytes)...  
Received: 13142141/13142141 bytes (100.0%)  
File transfer complete

**Сервер**

Accepted connection from 127.0.0.1:47430 (fd=4)  
Received file request from fd=4 for: baseball-cap-collection.zip  
Sending file response to fd=4: status=0, file\_size=13142141  
Client fd=4 is ready to receive the file  
Client fd=4: Sent 13142141/13142141 bytes (100%)  
File transfer complete for client fd=4  
Closed connection for client fd=4

### Робота з декількома клієнтами

**Скрипт для одночасного запуску 5-ти клієнтів**

#!/bin/bash  
  
./client 127.0.0.1 12345 AnyDesk.exe 100000000000000 &  
./client 127.0.0.1 12345 FigmaSetup.exe 100000000000000 &  
./client 127.0.0.1 12345 baseball-cap-collection.zip 100000000000000 &  
./client 127.0.0.1 12345 junos-vsrx-12.1X47-D10.4-domestic.ova 100000000000000 &  
./client 127.0.0.1 12345 n3467.pdf 1000 &

**Клієнт**

dymytryke@dymytryke:/mnt/c/Users/dymyt/OneDrive/Документы/save/Учеба/4\_курс/Netw.UNIX/4/bin$ Received response: protocol version 1, message type 2, status 0, file size 13142141  
Sending ready to receive message  
Received: 512/5634880 bytes (0.0%)Receiving file content (13142141 bytes)...  
Received response: protocol version 1, message type 2, status 0, file size 118362136  
Sending ready to receive message  
Received: 8704/5634880 bytes (0.2%)Receiving file content (118362136 bytes)...  
Received: 5634880/5634880 bytes (100.0%)  
File transfer complete  
Received: 5635584/13142141 bytes (42.9%)Received response: protocol version 1, message type 2, status 0, file size 238397440  
Sending ready to receive message  
Received: 5636096/118362136 bytes (4.8%)Receiving file content (238397440 bytes)...  
Received: 13142141/13142141 bytes (100.0%)  
File transfer complete  
Received: 13143040/118362136 bytes (11.1%)Received response: protocol version 1, message type 2, status 0, file size 4309251  
File size (4309251 bytes) exceeds maximum allowed size (1000 bytes)  
Received: 118362136/118362136 bytes (100.0%)  
File transfer complete  
Received: 238397440/238397440 bytes (100.0%)  
File transfer complete

**Сервер**

dymytryke@dymytryke:/mnt/c/Users/dymyt/OneDrive/Документы/save/Учеба/4\_курс/Netw.UNIX/4/bin$ stdbuf -oL ./multiplexing\_server 127.0.0.1 12345 ../test\_files/ 3 | ts  
Mar 25 08:49:48 Multiplexing server started at 127.0.0.1:12345, serving files from '../test\_files/', max clients: 3  
Mar 25 08:49:52 Accepted connection from 127.0.0.1:43086 (fd=4)  
Mar 25 08:49:52 Accepted connection from 127.0.0.1:43082 (fd=5)  
Mar 25 08:49:52 Received file request from fd=4 for: AnyDesk.exe  
Mar 25 08:49:52 Sending file response to fd=4: status=0, file\_size=5634880  
Mar 25 08:49:52 Accepted connection from 127.0.0.1:43084 (fd=6)  
Mar 25 08:49:52 Received file request from fd=5 for: baseball-cap-collection.zip  
Mar 25 08:49:52 Sending file response to fd=5: status=0, file\_size=13142141  
Mar 25 08:49:52 Client fd=4 is ready to receive the file  
Mar 25 08:49:52 Received file request from fd=6 for: FigmaSetup.exe  
Mar 25 08:49:52 Sending file response to fd=6: status=0, file\_size=118362136  
Mar 25 08:49:52 Client fd=5 is ready to receive the file  
Mar 25 08:49:52 Client fd=6 is ready to receive the file  
Client fd=4: Sent 5634880/5634880 bytes (100%)  
Mar 25 08:49:56 File transfer complete for client fd=4  
Mar 25 08:49:56 Closed connection for client fd=4  
Mar 25 08:49:56 Accepted connection from 127.0.0.1:43104 (fd=4)  
Mar 25 08:49:56 Received file request from fd=4 for: junos-vsrx-12.1X47-D10.4-domestic.ova  
Mar 25 08:49:56 Sending file response to fd=4: status=0, file\_size=238397440  
Mar 25 08:49:56 Client fd=4 is ready to receive the file  
Client fd=5: Sent 13142141/13142141 bytes (100%)  
Mar 25 08:50:03 File transfer complete for client fd=5  
Mar 25 08:50:03 Closed connection for client fd=5  
Mar 25 08:50:03 Accepted connection from 127.0.0.1:43112 (fd=5)  
Mar 25 08:50:03 Received file request from fd=5 for: n3467.pdf  
Mar 25 08:50:03 Sending file response to fd=5: status=0, file\_size=4309251  
Mar 25 08:50:03 Client fd=5 refused to receive the file (too large)  
Mar 25 08:50:03 Closed connection for client fd=5  
Client fd=6: Sent 118362136/118362136 bytes (100%)  
Mar 25 08:50:58 File transfer complete for client fd=6  
Mar 25 08:50:58 Closed connection for client fd=6  
Client fd=4: Sent 238397440/238397440 bytes (100%)  
Mar 25 08:51:25 File transfer complete for client fd=4  
Mar 25 08:51:25 Closed connection for client fd=4

**Demo:**

[Video](https://github.com/dymytryke/Netw.UNIX/blob/lab4/4/docs/multiple_clients_demo.mp4)