Московский Авиационный Институт (Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики Кафедра вычислительной математики и программирования

## Курсовой проект по курсу

**«Операционные системы»**

Группа: М80-207Б-20

Студент: Соколов Д.В. Преподаватель: Миронов Е.С. Оценка:

Дата: 04.03.22

Москва, 2022.

# Постановка задачи

Консоль-сервер сообщений. Необходимо написать консоль-серверное приложение с передачей между клиентами сообщений при этом предусмотреть создание групповых чатов. Необходимо написать 2 программы: сервер и клиент. Сначала запускается сервер, а далее клиенты соединяются с сервером. Сервер координирует клиентов между собой. При запуске клиента нужно ввести логин по которому будут передаваться сообщений:

* Подключение клиента с помощью логина
* Клиент может отправить сообщение другому клиенту по его логину
* Клиент в реальном времени принимает сообщения от других клиентов

**Вариант задания:**

Необходимо предусмотреть возможность создания «групповых чатов». Связь между сервером и клиентом должна быть реализована при помощи pipe'ов (именованных каналов).

# Алгоритм решения задачи

Программы я писал на языке Си, в стандарте POSIX есть встроенный FIFO-каналы. Для начала я написал модуль vector.c для динамических массивов, хранящих строку, чтобы было проще передавать и вводить сообщения. Моё приложение состоит из server и clients частей. Запуск сервера создаёт каналы через мои обёртки, так как при одностороннем открытии канала программа блокируется, то сервер ждёт присоединения клиентов:

CreatePipe("file0", 0666);

printf("Ok\n");

fd\_read = ConnectPipe("file0", O\_RDONLY);

printf("Server set up!\n"***);***

Сервер общается с клиентами посредством канала file0, именно с этого канала сервер читает постоянно данные, поэтому каждый клиент автоматически подключается к файлу file0. Я спроектировал сервер, который при добавлении клиента (подключение через команду login ‘name’ к серверу), сервер создаёт новый процесс с помощью fork() (POSIX), новый процесс запускает исполняющий файл server и посредством общего канала связываются в топологию ‘список’ (каждый новый процесс сервер, подключается к последнему с помощью канала):

if (!strncmp("/login", command, 7)) {

if (!child) {

memcpy(namefile, "file", strlen("file"));

strncat(namefile, to\_string(count), strlen(to\_string(count)) + 1);

count++;

CreatePipe(namefile, 0666);

CreateNewServer(namefile, name);

fd\_give = ConnectPipe(namefile, O\_WRONLY);

memcpy(Nameof, namefile, PIPE\_BUF);

child = 1;

} else {

memcpy(namefile, "file", strlen("file"));

strncat(namefile, to\_string(count), strlen(to\_string(count)) + 1);

count++;

strncat(buffer, " ", 2);

strncat(buffer, namefile, strlen(namefile) + 1);

write(fd\_give, buffer, strlen(buffer));

}

переменная child показывает, последний ли это процесс в списке.

Итак, для каждого клиента создаётся такой процесс, он нужен чтобы общаться с клиентом асинхронно. Клиент в свою очередь при исполнении команды login “форкает” новый процесс, и связывает серверный процесс:

printf("Login: ");

cmd = s\_gets();

printf("--------------------\n");

printf("%s\n", cmd->data);

memcpy(buffer, cmd->data, strlen(cmd->data) + 1);

if (CorrectWord(buffer)) {

SplitString(command, name, buffer);

} else {printf("Error: pls log in\n"); Destroy(cmd); continue;}

if (!strncmp(command, "login", 6)) {

memcpy(tmp, "/", 2);

strncat(tmp, buffer, strlen(buffer) + 1);

CreateFork(name);

memcpy(NodeName, name, strlen(name) + 1);

write(fd\_write, tmp, PIPE\_BUF);

Destroy(cmd);

break;

} else {printf("Error: pls log in\n"); Destroy(cmd); continue;}

SplitString() разделяет ввод на первое слово и остальную часть

CorrectWord() проверяет словосочетание в введунной строке

В итоге получается хорошая спроектированная система. Сначала сервер создает новый процесс для отправки ответов клиенту, а клиент раздваевается на два процесса - первый связан с головой списка сервера и вводит данные на сервер, а второй связывается с только что созданным процессом от сервера, получает данные от него и выводит их в консоль.

int CreateNewServer(char\* namefile, char\* pipename) {

int NodePid;

NodePid = fork();

if (NodePid == -1) {

printf("Error: fail fork()\n");

return -1;

} else if (NodePid == 0)

execl("./server", "server", namefile, pipename, NULL);

else printf("Ok: %d\n", NodePid);

printf("--------------------\n");

return 0;

}

void CreateFork(char\* name) {

int NodePid, fd\_read;

char buffer[PIPE\_BUF];

NodePid = fork();

if (NodePid == -1) {

printf("Error: fail fork()\n");

return;

} else if (NodePid == 0) {

CreatePipe(name, 0666);

fd\_read = ConnectPipe(name, O\_RDONLY);

while(1) {

ReadPipe(fd\_read, buffer);

printf("%s\n", buffer);

}

} //else printf("Ok: %d\n", NodePid);

}

В итоге получается получается асинхронная модель с мгновенными отправкой и получением сообщений, работающая асинхронно.

Вторым пунктом была отправка сообщений, тут всё просто, на клиентской части при вводе сообщения к нему добавляется логин отправителя, который был сохранйн в прошлом пункте. В итоге серверная часть проходит по всему списку и сравнивает имя кому нужно отправить с именем, находящимся на срвере. Если имя не совпало, то всё сообщение передается вниз по списку:

if (!strncmp(argv[2], Namekyda, strlen(Namekyda) + 1)) {

if (CorrectWord(other)) {

SplitString(Nameof, messegersd, other);

printf("--> %s <--\n", Nameof);

} else {

printf("No Message!\n");

continue;

}

memcpy(new\_msg, "Message: \n", 11);

strncat(new\_msg, Nameof, strlen(Nameof) + 1);

strncat(new\_msg, ": ", 3);

strncat(new\_msg, messegersd, strlen(messegersd) + 1);

printf("Message: \n"

"%s: %s\n", Nameof, messegersd);

write(fd\_write, new\_msg, PIPE\_BUF);

memset(new\_msg, '\0', PIPE\_BUF);

} else if (child) {

printf("--->\n");

write(fd\_give, buffer, PIPE\_BUF);

} else {

if (CorrectWord(other)) {

SplitString(Nameof, messegersd, other);

printf("--> %s <--\n", Nameof);

} else {

printf("No Message!\n");

continue;

}

printf("Not found %s\n", Nameof);

fd\_tmp = ConnectPipe(Nameof, O\_WRONLY);

printf("Not found\n");

write(fd\_tmp, "Server: Not found\n", PIPE\_BUF);

close(fd\_tmp);

memset(Nameof, '\0', PIPE\_BUF);

}

При нахождении клиенета в списке, ему по средством дескриптора передаётся сообщение, которое было соединено в едино с помощью строенных функция strncat().

Третий пункт, создание чата. Чат создаётся ещё проще, мы храним сообщение, клиентов и самого автора чата, также как и в первом пункте мы создаём процесс для новго элемента в списке, которые будет отвечать за чат. Но с одним условием, что в аргументе укажем ключа “-f”. Этот клиент устанавливает связь со всеми клиентами и заносит названия клиента и дескриптор в два вектора. Получается с помощью команды msg мы можем указывать и название чата, только нужно сделать проверку, чтобы автор сообщения не порлучал своё продублированное сообщение себе в консоль:

else if (!strncmp("/create", command, 5)) {

if (!child) {

if (CorrectWord(name))

SplitString(namechat, othername, name);

else {

printf("No Message---!\n");

continue;

}

CreatePipe(namechat, 0666);

CreateNewChat(namechat, othername);

fd\_give = ConnectPipe(namechat, O\_WRONLY);

child = 1;

} else {

write(fd\_give, buffer, PIPE\_BUF);

}

}

int CreateNewChat(char\* namefile, char\* pipename) {

int NodePid;

NodePid = fork();

if (NodePid == -1) {

printf("Error: fail fork()\n");

return -1;

} else if (NodePid == 0)

execl("./server", "server", namefile, pipename, "-f", NULL);

else printf("Ok: %d\n", NodePid);

printf("--------------------\n");

return 0;

}

Этот ключ помогает сделать аргументов 4, а следовательно подключаемые каналы будут записываться по-другому - в вектор.

fd\_read = ConnectPipe(argv[1], O\_RDONLY);

printf("%s startup--> %s\n", argv[1], argv[2]);

\_\_fd\_write = (vector\_int \*) malloc(sizeof(vector\_int));

CreateVec\_int(\_\_fd\_write, 2);

memcpy(tmpchat, argv[2], szof(argv[2]) + 1);

printf(":%s -- %d\n", tmpchat, szof(argv[2]));

while (CorrectWord(tmpchat)) {

printf("tyt1\n");

SplitString(argv1, argv2, tmpchat);

printf("%s\n", argv2);

if (indicator == 0) {

memcpy(Nameof, argv1, strlen(argv1) + 1);

}

if ((corrected = fopen(argv1, "rw")) != NULL) {

if (indicator >= \_\_fd\_write->size)

Resize\_int(\_\_fd\_write, \_\_fd\_write->size \* 2);

h[indicator] = (char \*) malloc(sizeof(char) \* PIPE\_BUF);

memcpy(h[indicator], argv1, strlen(argv1) + 1);

\_\_fd\_write->data[indicator] = ConnectPipe(argv1, O\_WRONLY);

indicator++;

memset(tmpchat, '\0', PIPE\_BUF);

memcpy(tmpchat, argv2, strlen(argv2) + 1);

fclose(corrected);

} else {

fd\_tmp = ConnectPipe(Nameof, O\_WRONLY);

printf("Not found\n");

memcpy(\_\_tmp, "Server: Not found ", strlen("Server: Not found ") + 1);

strncat(\_\_tmp, argv1, strlen(argv1) + 1);

write(fd\_tmp, \_\_tmp, PIPE\_BUF);

memset(tmpchat, '\0', PIPE\_BUF);

memcpy(tmpchat, argv2, strlen(argv2) + 1);

close(fd\_tmp);

}

так как во втором аргументе я передаю строку со всеми именами клиентов, я делю строку на слова, пока не останется одно слово. Проверяю можно ли открыть канал с помощью fopen(), чтобы сервер не блокировался, если такого канала нет, тогда он будет ждать открытия этого канала на другой стороне. Естественно рассылка сообщений чату идёт с помощью вектора:

for (int i = 0; i < indicator; i++)

if (strncmp(h[i], Nameof, strlen(h[i]) + 1))

write(\_\_fd\_write->data[i], new\_msg, PIPE\_BUF);

# Код программы

## client.c

#include <stdio.h>

#include <string.h>

#include "msg.h"

#include "vector.h"

#include "funcom.h"

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <limits.h>

int main(void) {

int fd\_write, fd\_read;

char buffer[PIPE\_BUF], command[PIPE\_BUF], name[PIPE\_BUF], other[PIPE\_BUF], tmp[PIPE\_BUF], NodeName[PIPE\_BUF], msg[PIPE\_BUF];

char namechat[PIPE\_BUF], tmpchat[PIPE\_BUF], othername[PIPE\_BUF];

char\* bufin; char\* msgin; char\* newstr;

vector\* cmd;

fd\_write = ConnectPipe("file0", O\_WRONLY);

printf("Connected in server\n");

while(1) {

printf("Login: ");

cmd = s\_gets();

printf("--------------------\n");

printf("%s\n", cmd->data);

memcpy(buffer, cmd->data, strlen(cmd->data) + 1);

if (CorrectWord(buffer)) {

SplitString(command, name, buffer);

} else {printf("Error: pls log in\n"); Destroy(cmd); continue;}

if (!strncmp(command, "login", 6)) {

memcpy(tmp, "/", 2);

strncat(tmp, buffer, strlen(buffer) + 1);

CreateFork(name);

memcpy(NodeName, name, strlen(name) + 1);

write(fd\_write, tmp, PIPE\_BUF);

Destroy(cmd);

break;

} else {printf("Error: pls log in\n"); Destroy(cmd); continue;}

Destroy(cmd);

}

while (1) {

cmd = s\_gets();

memcpy(buffer, cmd->data, strlen(cmd->data) + 1);

if (CorrectWord(buffer)) {

SplitString(command, name, buffer);

} else {printf("Incorrected command\n"); Destroy(cmd); continue;}

if (!strncmp(command, "msg", 4)) {

memcpy(tmp, "/", 2);

strncat(tmp, buffer, strlen(buffer) + 1);

strncat(tmp, " ", 2);

strncat(tmp, NodeName, strlen(NodeName) + 1);

printf("Message: ");

strncat(tmp, " ", 2);

msgin = fgets(msg, PIPE\_BUF, stdin);

newstr = strchr(msgin, '\n');

\*newstr = '\0';

printf("--------------------\n");

strncat(tmp, msgin, strlen(msgin) + 1);

write(fd\_write, tmp, PIPE\_BUF);

} else if (!strncmp(command, "create", 7)) {

msgin = fgets(msg, PIPE\_BUF, stdin);

newstr = strchr(msgin, '\n');

\*newstr = '\0';

memcpy(tmp, "/", 2);

strncat(tmp, buffer, strlen(buffer) + 1);

strncat(tmp, " ", 2);

strncat(tmp, NodeName, strlen(NodeName) + 1);

strncat(tmp, " ", 2);

strncat(tmp, msgin, strlen(msgin) + 1);

write(fd\_write, tmp, PIPE\_BUF);

} else {printf("--------------------\n"); printf("Incorrected command\n"); Destroy(cmd); continue;}

memset(buffer, '\0', PIPE\_BUF);

memset(namechat, '\0', PIPE\_BUF);

memset(tmpchat, '\0', PIPE\_BUF);

memset(othername, '\0', PIPE\_BUF);

memset(name, '\0', PIPE\_BUF);

memset(tmp, '\0', PIPE\_BUF);

Destroy(cmd);

}

return 0;

}

## server.c

#include <stdio.h>

#include "vector.h"

#include "funcom.h"

#include <stdlib.h>

#include "msg.h"

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <limits.h>

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

int NodePid = 0;

int fd\_read, fd\_write, fd\_give, child = 0, count = 1, fd\_tmp;

char buffer[PIPE\_BUF], command[PIPE\_BUF], name[PIPE\_BUF], other[PIPE\_BUF], new\_msg[PIPE\_BUF];

char namefile[PIPE\_BUF], namepid[PIPE\_BUF], Nameof[PIPE\_BUF], Namekyda[PIPE\_BUF], messegersd[PIPE\_BUF];

char namechat[PIPE\_BUF], tmpchat[PIPE\_BUF], othername[PIPE\_BUF], argv1[PIPE\_BUF], argv2[PIPE\_BUF];

char \_\_tmp[PIPE\_BUF];

int indicator = 0;

FILE\* corrected;

char\* h[PIPE\_BUF];

vector\_int\* \_\_fd\_write = NULL;

if (argc == 1) {

CreatePipe("file0", 0666);

printf("Ok\n");

fd\_read = ConnectPipe("file0", O\_RDONLY);

printf("Server set up!\n");

while (1) {

memset(buffer, '\0', PIPE\_BUF);

ReadPipe(fd\_read, buffer);

printf("%s\n", buffer);

if (CorrectWord(buffer))

SplitString(command, name, buffer);

else {printf("nocorrected\n"); continue;}

if (!strncmp("/login", command, 7)) {

if (!child) {

memcpy(namefile, "file", strlen("file"));

strncat(namefile, to\_string(count), strlen(to\_string(count)) + 1);

count++;

CreatePipe(namefile, 0666);

CreateNewServer(namefile, name);

fd\_give = ConnectPipe(namefile, O\_WRONLY);

memcpy(Nameof, namefile, PIPE\_BUF);

child = 1;

} else {

memcpy(namefile, "file", strlen("file"));

strncat(namefile, to\_string(count), strlen(to\_string(count)) + 1);

count++;

strncat(buffer, " ", 2);

strncat(buffer, namefile, strlen(namefile) + 1);

write(fd\_give, buffer, strlen(buffer));

}

} else if (!strncmp("/msg", command, 5)) {

if (child) {

write(fd\_give, buffer, strlen(buffer));

} else {

fd\_write = ConnectPipe(Nameof, O\_RDONLY);

write(fd\_write, "Server: Not found", strlen("Not found") + 1);

close(fd\_write);

}

} else if (!strncmp("/create", command, 5)) {

if (child) {

write(fd\_give, buffer, strlen(buffer));

} else {

fd\_write = ConnectPipe(Nameof, O\_RDONLY);

write(fd\_write, "Server: Not found", strlen("Not found") + 1);

close(fd\_write);

}

}

}

close(fd\_read);

} else {

if (argc < 4) {

fd\_read = ConnectPipe(argv[1], O\_RDONLY);

fd\_write = ConnectPipe(argv[2], O\_WRONLY);

} else {

fd\_read = ConnectPipe(argv[1], O\_RDONLY);

printf("%s startup--> %s\n", argv[1], argv[2]);

\_\_fd\_write = (vector\_int \*) malloc(sizeof(vector\_int));

CreateVec\_int(\_\_fd\_write, 2);

memcpy(tmpchat, argv[2], szof(argv[2]) + 1);

printf(":%s -- %d\n", tmpchat, szof(argv[2]));

while (CorrectWord(tmpchat)) {

printf("tyt1\n");

SplitString(argv1, argv2, tmpchat);

printf("%s\n", argv2);

if (indicator == 0) {

memcpy(Nameof, argv1, strlen(argv1) + 1);

}

if ((corrected = fopen(argv1, "rw")) != NULL) {

if (indicator >= \_\_fd\_write->size)

Resize\_int(\_\_fd\_write, \_\_fd\_write->size \* 2);

h[indicator] = (char \*) malloc(sizeof(char) \* PIPE\_BUF);

memcpy(h[indicator], argv1, strlen(argv1) + 1);

\_\_fd\_write->data[indicator] = ConnectPipe(argv1, O\_WRONLY);

indicator++;

memset(tmpchat, '\0', PIPE\_BUF);

memcpy(tmpchat, argv2, strlen(argv2) + 1);

fclose(corrected);

} else {

fd\_tmp = ConnectPipe(Nameof, O\_WRONLY);

printf("Not found\n");

memcpy(\_\_tmp, "Server: Not found ", strlen("Server: Not found ") + 1);

strncat(\_\_tmp, argv1, strlen(argv1) + 1);

write(fd\_tmp, \_\_tmp, PIPE\_BUF);

memset(tmpchat, '\0', PIPE\_BUF);

memcpy(tmpchat, argv2, strlen(argv2) + 1);

close(fd\_tmp);

}

}

printf("tyt2\n");

printf("is %s -- string\n", tmpchat);

if ((corrected = fopen(argv1, "rw")) != NULL) {

if (indicator >= \_\_fd\_write->size)

Resize\_int(\_\_fd\_write, \_\_fd\_write->size \* 2);

h[indicator] = (char \*) malloc(sizeof(char) \* PIPE\_BUF);

memcpy(h[indicator], argv1, strlen(argv1) + 1);

\_\_fd\_write->data[indicator] = ConnectPipe(tmpchat, O\_WRONLY);

indicator++;

printf("tyt\n");

for (int i = 0; i < indicator; i++)

printf("%d ", \_\_fd\_write->data[i]);

} else {

fd\_tmp = ConnectPipe(Nameof, O\_WRONLY);

printf("Not found\n");

memcpy(\_\_tmp, "Server: Not found ", strlen("Server: Not found ") + 1);

strncat(\_\_tmp, tmpchat, strlen(tmpchat) + 1);

write(fd\_tmp, \_\_tmp, PIPE\_BUF);

close(fd\_tmp);

}

memset(Nameof, '\0', PIPE\_BUF);

}

while (1) {

memset(buffer, '\0', PIPE\_BUF);

ReadPipe(fd\_read, buffer);

printf("%s\n", buffer);

if (CorrectWord(buffer))

SplitString(command, name, buffer);

else {printf("nocorrected\n"); continue;}

if (!strncmp("/login", command, 7)) {

SplitString(namepid, other, name);

if (!child) {

CreatePipe(other, 0666);

CreateNewServer(other, namepid);

fd\_give = ConnectPipe(other, O\_WRONLY);

child = 1;

} else {

write(fd\_give, buffer, strlen(buffer));

}

} else if (!strncmp("/msg", command, 5)) {

SplitString(Namekyda, other, name);

if (argc < 4) {

if (!strncmp(argv[2], Namekyda, strlen(Namekyda) + 1)) {

if (CorrectWord(other)) {

SplitString(Nameof, messegersd, other);

printf("--> %s <--\n", Nameof);

} else {

printf("No Message!\n");

continue;

}

memcpy(new\_msg, "Message: \n", 11);

strncat(new\_msg, Nameof, strlen(Nameof) + 1);

strncat(new\_msg, ": ", 3);

strncat(new\_msg, messegersd, strlen(messegersd) + 1);

printf("Message: \n"

"%s: %s\n", Nameof, messegersd);

write(fd\_write, new\_msg, PIPE\_BUF);

memset(new\_msg, '\0', PIPE\_BUF);

} else if (child) {

printf("--->\n");

write(fd\_give, buffer, PIPE\_BUF);

} else {

if (CorrectWord(other)) {

SplitString(Nameof, messegersd, other);

printf("--> %s <--\n", Nameof);

} else {

printf("No Message!\n");

continue;

}

printf("Not found %s\n", Nameof);

fd\_tmp = ConnectPipe(Nameof, O\_WRONLY);

printf("Not found\n");

write(fd\_tmp, "Server: Not found\n", PIPE\_BUF);

close(fd\_tmp);

memset(Nameof, '\0', PIPE\_BUF);

}

} else {

if (!strncmp(argv[1], Namekyda, strlen(Namekyda) + 1)) {

if (CorrectWord(other))

SplitString(Nameof, messegersd, other);

else {

printf("No Message!\n");

continue;

}

memcpy(new\_msg, "Message: \n", 11);

strncat(new\_msg, argv[1], strlen(argv[1]) + 1);

strncat(new\_msg, ":", 2);

strncat(new\_msg, Nameof, strlen(Nameof) + 1);

strncat(new\_msg, ": ", 3);

strncat(new\_msg, messegersd, strlen(messegersd) + 1);

printf("Message: \n"

"%s: %s\n", Nameof, messegersd);

for (int i = 0; i < indicator; i++)

if (strncmp(h[i], Nameof, strlen(h[i]) + 1))

write(\_\_fd\_write->data[i], new\_msg, PIPE\_BUF);

memset(new\_msg, '\0', PIPE\_BUF);

} else if (child) {

printf("--->\n");

write(fd\_give, buffer, PIPE\_BUF);

} else {

if (CorrectWord(other)) {

SplitString(Nameof, messegersd, other);

printf("--> %s <--\n", Nameof);

} else {

printf("No Message!\n");

continue;

}

printf("Not found %s\n", Nameof);

fd\_tmp = ConnectPipe(Nameof, O\_WRONLY);

printf("Not found\n");

write(fd\_tmp, "Server: Not found\n", PIPE\_BUF);

close(fd\_tmp);

memset(Nameof, '\0', PIPE\_BUF);

}

}

} else if (!strncmp("/create", command, 5)) {

if (!child) {

if (CorrectWord(name))

SplitString(namechat, othername, name);

else {

printf("No Message---!\n");

continue;

}

CreatePipe(namechat, 0666);

CreateNewChat(namechat, othername);

fd\_give = ConnectPipe(namechat, O\_WRONLY);

child = 1;

} else {

write(fd\_give, buffer, PIPE\_BUF);

}

}

memset(namechat, '\0', PIPE\_BUF);

memset(tmpchat, '\0', PIPE\_BUF);

memset(othername, '\0', PIPE\_BUF);

}

close(fd\_read);

close(fd\_write);

}

return 0;

}

## msg.c

#include "msg.h"

#include "vector.h"

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <limits.h>

void CreatePipe(char\* name, unsigned int user) {

unlink(name);

if (mkfifo(name, (mode\_t) user)) {

perror("mkfifo error:");

exit(EXIT\_FAILURE);

}

}

int ConnectPipe(char\* name, int mode) {

int fd\_info;

while ((fd\_info = open(name, mode, 0)) == -1)

continue;

return fd\_info;

}

static void clear(void) {

char c = getchar();

while (c != '\n' && c != '\0' && c != EOF)

c = getchar();

}

vector\* s\_gets(void) {

vector\* tmp = (vector \*) malloc(sizeof(vector));

CreateVec(tmp, 2);

char c;

int i = 0;

c = getchar();

while (c != ' ' && c != '\n') {

if (i >= tmp->size)

Resize(tmp, tmp->size \* 2);

Save(tmp, i, c);

i++;

c = getchar();

}

if (i >= tmp->size)

Resize(tmp, tmp->size \* 2);

Save(tmp, i, c);

i++;

c = getchar();

while (c != ' ' && c != '\n') {

if (i >= tmp->size)

Resize(tmp, tmp->size \* 2);

Save(tmp, i, c);

i++;

c = getchar();

}

if (c == ' ')

clear();

if (i >= tmp->size)

Resize(tmp, tmp->size \* 2);

Save(tmp, i, '\0');

i++;

tmp->size = i;

return tmp;

}

void SplitString(char\* command, char\* other, char\* arr) {

char\* start = arr;

char\* space = strchr(arr, ' ');

char\* end = strchr(arr, '\0');

memcpy(command, arr, space - start);

int diff = space - start;

\*(command + diff) = '\0';

while (\*space == ' ')

space++;

memcpy(other, space, end - space + 1);

}

int CorrectWord(char\* string) {

char\* chng = strchr(string, ' ');

return chng ? 1 : 0;

}

void ReadPipe(int \_\_fd, char\* buf) {

while (read(\_\_fd, buf, PIPE\_BUF) <= 0)

continue;

}

## funcom.c

#include "funcom.h"

#include <stdio.h>

#include <stdlib.h>

#include "msg.h"

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <limits.h>

int CreateNewServer(char\* namefile, char\* pipename) {

int NodePid;

NodePid = fork();

if (NodePid == -1) {

printf("Error: fail fork()\n");

return -1;

} else if (NodePid == 0)

execl("./server", "server", namefile, pipename, NULL);

else printf("Ok: %d\n", NodePid);

printf("--------------------\n");

return 0;

}

int CreateNewChat(char\* namefile, char\* pipename) {

int NodePid;

NodePid = fork();

if (NodePid == -1) {

printf("Error: fail fork()\n");

return -1;

} else if (NodePid == 0)

execl("./server", "server", namefile, pipename, "-f", NULL);

else printf("Ok: %d\n", NodePid);

printf("--------------------\n");

return 0;

}

void CreateFork(char\* name) {

int NodePid, fd\_read;

char buffer[PIPE\_BUF];

NodePid = fork();

if (NodePid == -1) {

printf("Error: fail fork()\n");

return;

} else if (NodePid == 0) {

CreatePipe(name, 0666);

fd\_read = ConnectPipe(name, O\_RDONLY);

while(1) {

ReadPipe(fd\_read, buffer);

printf("%s\n", buffer);

}

} //else printf("Ok: %d\n", NodePid);

}

unsigned long szof(char\* string) {

unsigned long i = 0;

char\* start = string;

while (\*start != '\0') {

start++;

i++;

}

return i;

}

## vector.c

#include "vector.h"

#include <stdlib.h>

#include <stdio.h>

#include <ctype.h>

#include <string.h>

void CreateVec(vector\* v, int sz) {

v->size = sz;

v->data = (char \*) malloc(sizeof(char) \* sz);

}

int Empty(vector\* v) {

return v->size == 0 ? 1 : 0;

}

char Load(vector\* v, int i) {

return (i >= 0 && i < v->size) ? v->data[i] : -1;

}

void Resize(vector\* v, int sz) {

v->data = (char \*) realloc(v->data, sizeof(char) \* sz);

v->size = sz;

}

void Save(vector\* v, int i, char ch) {

i >= 0 && i < v->size ? v->data[i] = ch : 0;

}

void Destroy(vector\* v) {

v->size = 0;

free(v->data);

}

void CreateVec\_int(vector\_int\* v, int sz) {

v->size = sz;

v->data = (int \*) malloc(sizeof(int) \* sz);

}

int Empty\_int(vector\_int\* v) {

return v->size == 0 ? 1 : 0;

}

int Load\_int(vector\_int\* v, int i) {

return (i >= 0 && i < v->size) ? v->data[i] : -1;

}

void Resize\_int(vector\_int\* v, int sz) {

v->data = (int \*) realloc(v->data, sizeof(int) \* sz);

v->size = sz;

}

void Save\_int(vector\_int\* v, int i, int ch) {

i >= 0 && i < v->size ? v->data[i] = ch : 0;

}

void Destroy\_int(vector\_int\* v) {

v->size = 0;

free(v->data);

}

void CopyVec(vector\* src, vector\* out, int size) {

char tmp = -1;

int i = 0;

int j = size;

CreateVec(out, size);

tmp = Load(src, j);

while (!isdigit(tmp)) {

j--;

tmp = Load(src, j);

}

while (j >= 0) {

tmp = Load(src, j);

Save(out, i, tmp);

j--;

i++;

}

Save(out, i, '\0');

}

char\* to\_string(int number) {

vector str;

vector new;

int tmp, i = 0;

char ch;

CreateVec(&str, 2);

if (!number) {

Save(&str, i, '0');

i++;

}

while (number) {

tmp = number % 10;

ch = tmp + 48;

if (i >= str.size)

Resize(&str, str.size \* 2);

Save(&str, i, ch);

number /= 10;

i++;

}

if (i >= str.size)

Resize(&str, str.size \* 2);

Save(&str, i, '\0');

CopyVec(&str, &new, str.size);

return new.data;

}

# client1:

danchik@danchik-MS-7A20:~/Рабочий стол/repositories/Lab\_OOP/OS/Kp$ ./client

Connected in server

Login: login monster

--------------------

login monster

Message:

kostya: Welcome to Hell :D

msg kostya

Message: Lmao)

--------------------

Message:

chatik:heroes: Hello world!

Message:

chatik:kostya: hi)

msg chatik

Message: xDD

--------------------

msg chatik

Message: this work... I shoked

--------------------

Message:

heroes: remove kostya this chat pls...

msg heroes

Message: Net :DDDDD

--------------------

# client2:

danchik@danchik-MS-7A20:~/Рабочий стол/repositories/Lab\_OOP/OS/Kp$ ./client

Connected in server

Login: login heroes

--------------------

login heroes

msg noname

Message: PRIVET

--------------------

Server: Not found

create chatik

monster kostya

msg chatik

Message: Hello world!

--------------------

Message:

chatik:kostya: hi)

# Результат работы

Message:

chatik:monster: xDD

Message:

chatik:monster: this work... I shoked

msg monster

Message: remove kostya this chat pls...

--------------------

Message:

monster: Net :DDDDD

# 

# client3:

danchik@danchik-MS-7A20:~/Рабочий стол/repositories/Lab\_OOP/OS/Kp$ ./client

Connected in server

Login: login kostya

--------------------

login kostya

msg monster

Message: Welcome to Hell :D

--------------------

Message:

monster: Lmao)

Message:

chatik:heroes: Hello world!

msg chatik

Message: hi)

--------------------

Message:

chatik:monster: xDD

Message:

chatik:monster: this work... I shoked

# Вывод

В курсовом проекте мне была предложена интересная задача разработки консоль-серверного приложения сервера сообщений. Во время разработки возникло много проблем: как составить топологию, как заставить клиентов общаться, как принимать и отправлять мгновенные сообщения и т.д. Перед тем как программировать я спроектировал набросок задания на бумаге и почти все вопросы отпали. На этапе интенсивной разработки самое сложно было отлов ошибок из-за чего приложение падало. Задание получилось интересным и познавательным, хоть именованные каналы не так часто используются как сокеты и очереди сообщений.