|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

*ФАКУЛЬТЕТ «Информатика и системы управления»*

*КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»*

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** |  |

**Дисциплина:  *Операционные системы***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ***ИУ7И-66Б*** |  |  | **Нгуен Ф. С.** |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | **Рязанова Н. Ю.** |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

*Москва, 2021*

**I. Первая часть (AF\_UNIX)**

**Server.c**

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <signal.h>

#include <sys/types.h>

#include <sys/socket.h>

#define SOCKET\_NAME "./socket"

#define BUF\_SIZE 256

#define OK 0

static int sockfd;

void cleanup\_socket(void)

{

close(sockfd);

unlink(SOCKET\_NAME);

}

void sigint\_handler(int signum)

{

cleanup\_socket();

exit(OK);

}

int main(void)

{

if ((sockfd = socket(AF\_UNIX, SOCK\_DGRAM, 0)) < 0)

{

perror("Failed to create socket");

return EXIT\_FAILURE;

}

struct sockaddr srvr\_name;

srvr\_name.sa\_family = AF\_UNIX;

strcpy(srvr\_name.sa\_data, SOCKET\_NAME);

if (bind(sockfd, &srvr\_name, strlen(srvr\_name.sa\_data) + sizeof(srvr\_name.sa\_family)) < 0)

{

perror("Failed to bind socket");

return EXIT\_FAILURE;

}

signal(SIGINT, sigint\_handler);

fprintf(stdout, "Server is listening.\nTo stop server press Ctrl + C.\n");

char buf[BUF\_SIZE];

for (;;)

{

int bytes = recv(sockfd, buf, sizeof(buf), 0);

if (bytes <= 0)

{

perror("Failed to recv");

cleanup\_socket();

return EXIT\_FAILURE;

}

buf[bytes] = '\0';

fprintf(stdout, "Server read: [%s]\n", buf);

}

fprintf(stdout, "Server stopped listening\n");

cleanup\_socket();

fprintf(stdout, "Socket closed\n");

return OK;

}

**Client.c**

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#define SOCKET\_NAME "./socket"

#define BUF\_SIZE 256

#define OK 0

int main(void)

{

int sockfd = socket(AF\_UNIX, SOCK\_DGRAM, 0);

if (sockfd < 0)

{

perror("Failed to create socket");

return EXIT\_FAILURE;

}

struct sockaddr srvr\_name;

srvr\_name.sa\_family = AF\_UNIX;

strcpy(srvr\_name.sa\_data, SOCKET\_NAME);

char buf[BUF\_SIZE];

snprintf(buf, BUF\_SIZE, "This Message From %d", getpid());

if (sendto(sockfd, buf, strlen(buf), 0, &srvr\_name, strlen(srvr\_name.sa\_data) + sizeof(srvr\_name.sa\_family)) < 0)

{

perror("Failed to send message");

close(sockfd);

return EXIT\_FAILURE;

}

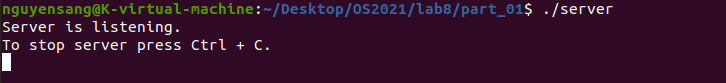
printf("Client sent: [%s]\n", buf);

return OK;

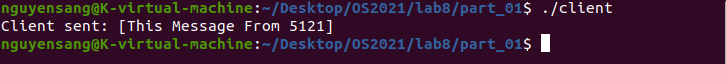
}

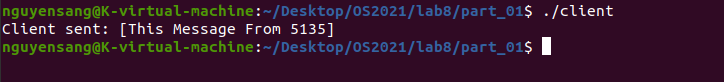
**Результат**

**$ ./server**

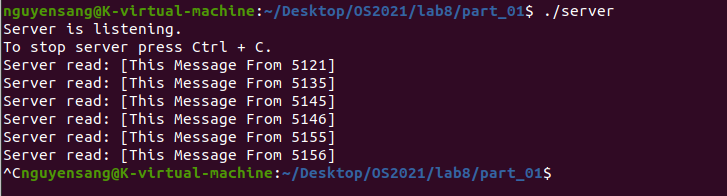


**$ ./Client**





…..



**Ii. Вторая часть (AF\_INET)**

**server.c**

#include "socket.h"

#define MAX\_CLIENTS\_COUNT 10

static int master\_sd;

static int clients[MAX\_CLIENTS\_COUNT];

int cleanup()

{

close(master\_sd);

exit(EXIT\_FAILURE);

}

**void sigint\_handler(int signum)**

{

cleanup();

exit(OK);

}

**void handle\_connection(void)**

{

const int sd = accept(master\_sd, NULL, NULL);

if (sd == -1) {

cleanup();

}

for (int i = 0; i < MAX\_CLIENTS\_COUNT; ++i)

{

if (!clients[i])

{

clients[i] = sd;

fprintf(stdout, "New connection.\n");

return;

}

}

fprintf(stderr, "Reached MAX\_CLIENTS\_COUNT (%d)\n", MAX\_CLIENTS\_COUNT);

cleanup();

}

**void handle\_client(int i)**

{

char msg[BUF\_SIZE];

const ssize\_t bytes = recv(clients[i], &msg, BUF\_SIZE, 0);

if (!bytes)

{

close(clients[i]);

clients[i] = 0;

return;

}

msg[bytes] = '\0';

fprintf(stdout, "Get message from client: %s\n", msg);

}

**int main(void)**

{

if ((master\_sd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror("Failed to create socket");

return EXIT\_FAILURE;

}

struct sockaddr\_in addr = {

.sin\_family = AF\_INET,

.sin\_addr.s\_addr = INADDR\_ANY,

.sin\_port = htons(SOCKET\_PORT)

};

if (bind(master\_sd, (struct sockaddr \*) &addr, sizeof addr) < 0)

{

cleanup();

}

if (listen(master\_sd, MAX\_CLIENTS\_COUNT) < 0)

{

cleanup();

}

signal(SIGINT, sigint\_handler);

fprintf(stdout, "Server is listening.\nTo stop server press Ctrl + C.\n");

while (1)

{

fd\_set readfds;

FD\_ZERO(&readfds);

FD\_SET(master\_sd, &readfds);

int max\_sd = master\_sd;

for (int i = 0; i < MAX\_CLIENTS\_COUNT; ++i)

{

if (clients[i] > 0)

{

FD\_SET(clients[i], &readfds);

}

if (clients[i] > max\_sd)

{

max\_sd = clients[i];

}

}

if (pselect(max\_sd + 1, &readfds, NULL, NULL, NULL, NULL) < 0)

{

cleanup();

perror("Failed to select");

}

if (FD\_ISSET(master\_sd, &readfds))

{

handle\_connection();

}

for (int i = 0; i < MAX\_CLIENTS\_COUNT; ++i)

{

if (clients[i] && FD\_ISSET(clients[i], &readfds))

{

handle\_client(i);

}

}

}

}

**Client.c**

include "socket.h"

**int main(void)**

{

const int master\_sd = socket(AF\_INET, SOCK\_STREAM, 0);

if (master\_sd == -1) {

perror("Failed to create socket");

return EXIT\_FAILURE;

}

struct sockaddr\_in addr = {

.sin\_family = AF\_INET,

.sin\_addr.s\_addr = INADDR\_ANY,

.sin\_port = htons(SOCKET\_PORT)

};

if (connect(master\_sd, (struct sockaddr \*) &addr, sizeof addr) < 0) {

perror("Failed to connect");

return EXIT\_FAILURE;

}

while (1)

{

char msg[BUF\_SIZE];

snprintf(msg, BUF\_SIZE, "My pid is %d", getpid());

if (sendto(master\_sd, msg, strlen(msg), 0, (struct sockaddr \*) &addr, sizeof addr) < 0)

{

perror("Failed to sendto");

return EXIT\_FAILURE;

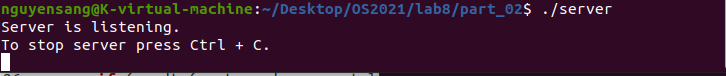
}

sleep(1);

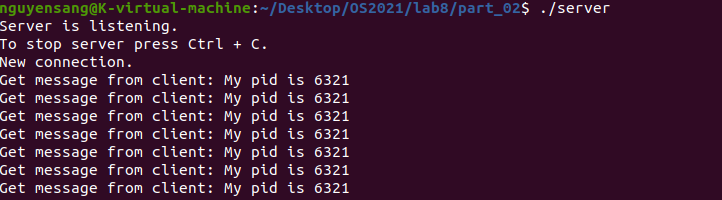
}

}

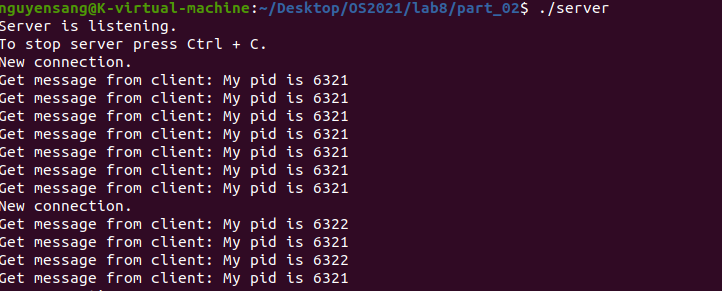
**$ ./server**

****

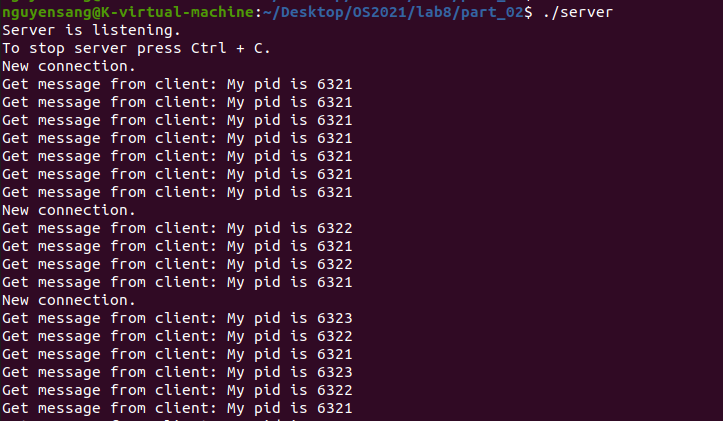
**$ ./client**

****

**$ ./client**

****

**$ ./client**

****