Министерство науки и высшего образования Российской Федерац
Федеральное государственное бюджетное образовательное учрежд
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»

Лабораторная работа № 6
Тема: сокеты
Дисциплина: Операционные системы

Студент Лучина Е.Д

Группа ИУ7-61Б

Преподаватель Рязанова Н.Ю.

Москва.
2020 г.

Задание 1

Организовать взаимодействие параллельных процессов на отдельном компьютере. Написать приложение по модели клиент-сервер, демонстрирующее взаимодействие параллельных процессов на отдельном компьютере с использованием сокетов в файловом пространстве имен: семейство - AF_UNIX, тип - SOCK_DGRAM.

server1.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/un.h>

#define SOCKET_NAME "mysocket.soc"
#define MSG_LEN 30

int sock;

void signal_handler(int signum)
{
        printf("\nclosing socket (ctrl+C)\n");
        close(sock);
        unlink(SOCKET_NAME);
        exit(0);
}

int main(void)
{
        struct sockaddr sock_addr;
        char msg[MSG_LEN];

        sock = socket(AF_UNIX, SOCK_DGRAM, 0);
```

```
        if (sock < 0)
        {
                perror("socket error\n");
                return sock;
        }

        sock_addr.sa_family = AF_UNIX;
        strcpy(sock_addr.sa_data, SOCKET_NAME);
        if (bind(sock, &sock_addr, sizeof(sock_addr)) < 0)
        {
                printf("closing socket\n");
                close(sock);
                unlink(SOCKET_NAME);
                perror("binding error\n");
                return -1;
        }
        printf("server waits for clients\n");
        signal(SIGINT, signal_handler);
        while(1)
        {
                int rec = recv(sock, msg, MSG_LEN, 0);
                if (rec < 0)
                {
                        printf("closing socket\n");
                        close(sock);
                        unlink(SOCKET_NAME);
                        perror("message receiving error\n");
                        return rec;
                }
                msg[rec] = 0;
                printf("Client message: %s\n", msg);
        }
}
```

client1.c

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

#define SOCKET_NAME "mysocket.soc"
#define MSG_LEN 30

int main(void)
{
        int sockfd = socket(AF_UNIX, SOCK_DGRAM, 0);
        if (sockfd < 0)
        {
                printf("error in socket()\n");
```

```
                return sockfd;
        }

        struct sockaddr server_addr;
        server_addr.sa_family = AF_UNIX;
        strcpy(server_addr.sa_data, SOCKET_NAME);

        char msg[MSG_LEN];
        printf("I am client %d\n", getpid());
        sprintf(msg, "hello from client %d\n", getpid());
        sendto(sockfd, msg, strlen(msg), 0, &server_addr, sizeof(server_addr));
        close(sockfd);
        printf("message is sent\n");
        return 0;
}
```

Пояснения к коду

С помощью функции socket() получаем дескриптор сокета. Первый параметр этой функции – домен (тип соединения), к которому принадлежит сокет. AF_UNIX, соответствует сокетам в файловом пространстве имен. Второй параметр SOCK_DGRAM определяет тип сокета как датаграммный. Третий параметр функции socket() позволяет указать протокол, используемый для передачи данных. В случае ошибки функция socket() возвращает -1.

Далее инициализируется структура socket_addr. Поле sa_family содержит указание семейства адресов, а sa_data - имя сокета.

После чего в программе-сервере вызывается функция bind(), которая связывает сокет с заданным адресом. Первым параметром функции является дескриптор, вторым – указатель на структуру sock_addr, третьим – длина структуры, содержащей адрес. Теперь программа-сервер становится доступна для соединения по заданному адресу (имени файла).

При обмене данными с сокетом используются функции recv() (в программе-сервере для получения сообщения) и sendto() (в программе-клиенте для отправки сообщения).

Функция recv() блокирует программу до тех пор, пока на входе не появятся новые данные. Первый аргумент - сокет-дескриптор, из которого читаются данные. Второй и третий аргументы - соответственно, адрес и длина буфера для записи читаемых данных. Четвертый параметр - это комбинация битовых флагов, управляющих режимами чтения. Аргумент flags равен нулю значит считанные данные удаляются из сокета. Функция возвращает число считанных байтов или -1 в случае ошибки.

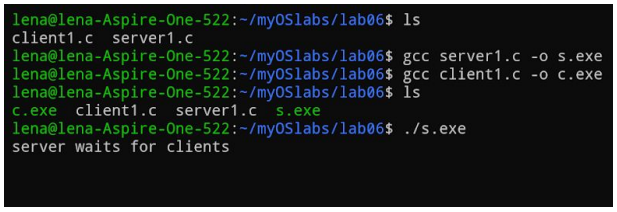
Программа-сервер считывает данные из сокета в бесконечном цикле. Завершается программа по сигналу SIGINT - функция signal_handler() «закрывает» сокет с помощью «файловой» функции close(). Функция unlink() удаляет файл сокета. Это же происходит и при ошибках связывания (binding error) или считывания сообщения (receiving message error).

В программе-клиенте вызов sprintf(msg, "hello from client %d\n", getpid()); формирует сообщение. А sendto(sockfd, msg, strlen(msg), 0, &server_addr, sizeof(server_addr)); его отправляет. Первый параметр функции sendto() – дескриптор сокета, второй и третий параметры позволяют указать адрес буфера для передачи данных и его длину. Четвертый параметр предназначен для передачи дополнительных флагов. Предпоследний и последний параметры несут информацию об адресе сервера и его длине, соответственно.

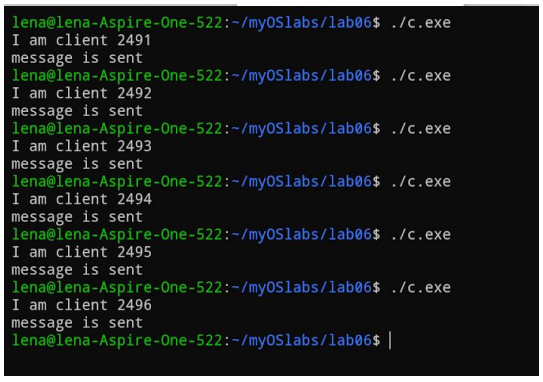
После окончания передачи данных сокет закрывается с помощью close().

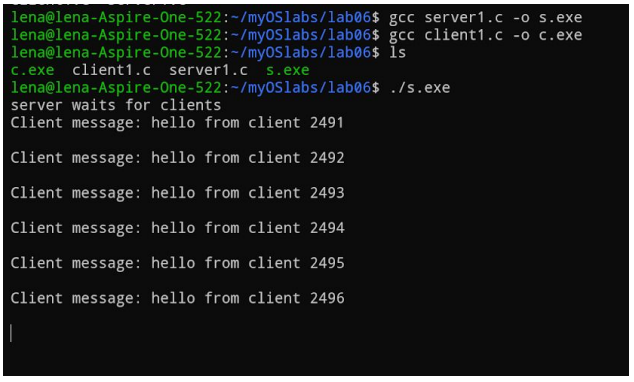
Результат работы

Скомпилируем программы и запустим сервер. В другом терминале можем посмотреть содержимое директории (ls -al) - там появится созданный сокет mysocket.soc.

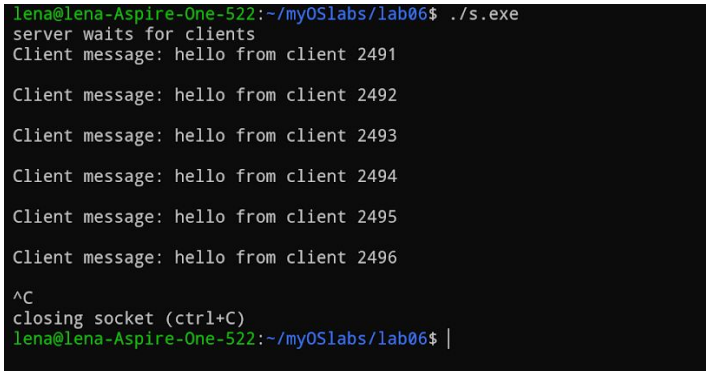


В другом терминале запустим несколько клиентов, каждый из которых будет выводить pid процесса и сообщение об осуществленной отправки. При этом в терминале, где запущен сервер, будем наблюдать получение этих сообщений.





Сигнал ctrl+c завершит работу сервера.



Задание 2

Организовать взаимодействие параллельных процессов в сети (ситуацию моделируем на одной машине). Написать приложение по модели клиент-сервер, осуществляющее

взаимодействие параллельных процессов, которые выполняются на разных компьютерах. Для взаимодействия с клиентами сервер должен использовать мультиплексирование. Сервер должен обслуживать запросы параллельно запущенных клиентов.

server2.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>

#define MAX_CLIENTS 10
#define MSG_LEN 30
#define SOCK_PORT 8088

int clients[MAX_CLIENTS] = {0};
int sock;

int connection_handling(int cur_fd)
{
        struct sockaddr_in addr;
        int addr_len = sizeof(addr);
        int fd;
        if ((fd = accept(cur_fd, (struct sockaddr *) &addr, (socklen_t *)
&addr_len)) < 0)
        {
                close(cur_fd);
                perror("accessing error\n");
                exit(-1);
        }

        printf("new connection: fd = %d\tip%s: %d\n", fd,
inet_ntoa(addr.sin_addr), ntohs(addr.sin_port));
        for (int i = 0; i < MAX_CLIENTS; i++)
        {
                if (clients[i] == 0)
                {
                        clients[i] = fd;
                        return 0;
                }
        }
        return -1;
}

void client_handling(int fd, int i)
{
        struct sockaddr_in addr;
        int addr_len = sizeof(addr);
```

```
        char msg[MSG_LEN];

        int rec = recv(fd, &msg, MSG_LEN, 0);
        if (rec <= 0)
        {
                getpeername(fd, (struct sockaddr*) &addr, (socklen_t *)
&addr_len);
                printf("client %d is disconnected %s: %d\n", i,
inet_ntoa(addr.sin_addr), ntohs(addr.sin_port));
                close(fd);
                clients[i] = 0;
        }
        else
        {
                msg[rec] = 0;
                printf("got message from client %d: %s\n", i + 1, msg);
        }
}

int main(void)
{
        sock = socket(AF_INET, SOCK_STREAM, 0);
        if (sock < 0)
        {
                perror("socket error\n");
                return -1;
        }
        struct sockaddr_in serv_sock;
        serv_sock.sin_family = AF_INET;
        serv_sock.sin_addr.s_addr = INADDR_ANY;
        serv_sock.sin_port = htons(SOCK_PORT);
        if (bind(sock, (struct sockaddr *) &serv_sock, sizeof(serv_sock)) < 0)
        {
                perror("binding error\n");
                close(sock);
                return -1;
        }

        if (listen (sock , 3) < 0)
        {
                perror("server can't listen\n");
                close(sock);
                return -1;
        }
        printf("server active on ip %s port %d\n", inet_ntoa(serv_sock.sin_addr),
ntohs(serv_sock.sin_port));
        while (1)
        {
                int max_fd = sock;
                fd_set set;
                FD_ZERO(&set);
                FD_SET(sock, &set);

                for (int i = 0; i < MAX_CLIENTS; i++)
                {
```

```
                        if (clients[i] > 0)
                                FD_SET(clients[i], &set);
                        if (clients[i] > max_fd)
                                max_fd = clients[i];
                }

                int active_clients_count = select(max_fd + 1, &set, NULL, NULL,
NULL);
                if (active_clients_count < 0)
                {
                        perror("there is no active clients\n");
                        close(sock);
                        return -1;
                }
                if (FD_ISSET(sock, &set))
                {
                        if (connection_handling(sock) < 0)
                        {
                                perror("connection error\n");
                                close(sock);
                                return -1;
                        }
                }
                for (int i = 0; i < MAX_CLIENTS; i++)
                {
                        int client = clients[i];
                        if (client > 0 && FD_ISSET(client, &set))
                                client_handling(client, i);
                }
        }
}
```

client2.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include <signal.h>
#include <string.h>

#define MAX_CLIENTS 10
#define MAX_MSG 5
#define MSG_LEN 30
#define SOCK_PORT 8088

int main(void)
```

```
{
        srand(time(NULL));

        int sock = socket(AF_INET, SOCK_STREAM, 0);
        if (sock < 0)
        {
                perror("socket error\n");
                return -1;
        }

        struct hostent* host = gethostbyname(SOCK_ADDR);
        if (!host)
        {
                perror("host not found\n");
                close(sock);
                return -1;
        }
        struct sockaddr_in serv_sock;
        serv_sock.sin_family = AF_INET;
        serv_sock.sin_addr = *((struct in_addr *)host->h_addr_list[0]);
        serv_sock.sin_port = htons(SOCK_PORT);

        if (connect(sock, (struct sockaddr *)&serv_sock, sizeof(serv_sock)) < 0)
        {
                perror("connecting error\n");
                close(sock);
                return -1;
        }

        char msg[MSG_LEN];
        for (int i = 0; i < MAX_MSG; i++)
        {
                sprintf(msg, "message #%d", i);
                if (send(sock, msg, strlen(msg), 0) < 0)
                {
                        perror("sending error\n");
                        close(sock);
                        return -1;
                }
                printf("msg #%d is sent\n", i);

                int wait_time = 1 + rand() % 5;
                sleep(wait_time);
        }

        printf("client work is done\n");
        close(sock);
        return 0;
}
```

Пояснения к коду

Вызовом `socket(AF_INET, SOCK_STREAM, 0);` создается сокет семейства AF_INET (сетевой) типа SOCK_STREAM (потоковый).

Далее инициализируется переменная типа структуры sockaddr_in, предназначеной для хранения адресов в формате Интернета. В качестве семейства адресов указывается AF_INET. Полю `sin_port` присваивается результат функции htons(), которая переписывает двухбайтовое значение порта так, чтобы порядок байтов соответствовал сетевому. А в качестве самого адреса указывается специальная константу INADDR_ANY, благодаря которой программа-сервер зарегистрируется на всех адресах той машины, на которой она
выполняется.

С помощью вызова bind() в программе-сервере сокет связывается с адресом. После этого сервер переводится в режим ожидания на системном вызове listen(), ожидая запроса на соединение. Максимальное число соединений, которые сервер может обрабатывать одновременно равно трем.

Функция select() проверяет состояние нескольких дескрипторов сокетов сразу. Первый параметр функции – количество проверяемых дескрипторов. Второй, третий и четвертый параметры функции представляют собой наборы дескрипторов, которые следует проверять, соответственно, на готовность к чтению, записи и на наличие исключительных ситуаций. Нас интересует только чтение (передаем сформированный раннее с помощью макроса FD_SET набор дескрипторов set, остальные аргументы - NULL). Сама функция select() – блокирующая, она возвращает управление, если хотя бы один из проверяемых сокетов готов к выполнению соответствующей операции.

Если есть данные для чтения вызывается функция connection_handing(), внутри которой с помощью вызова accept() устанавливается соединение в ответ на запрос клиента и создается копию сокета для того, чтобы исходный сокет мог продолжать прослушивание.

В цикле с помощью функции client_handling() осуществляется считывание `recv(fd, &msg, MSG_LEN, 0)` и вывод сообщения от клиента на сервере. Если recv() вернула ноль, то соединение было сброшено, выводится сообщение об отключении клиента и происходит закрытие сокета.

В процессе-клиенте функция gethostbyname() преобразует доменный адрес в сетевой, благодаря которому можно установить соединение. Соединение устанавливается вызовом connect(). Затем в цикле происходит отправка сообщений msg серверу `send(sock, msg, strlen(msg), 0)`. После завершения работы клиента сокет закрывается.

Результаты работы