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»

Лабораторная работа № 6
по операционным системам

Студент Колганов О.С.

Группа ИУ7 — 62Б

Преподаватель Рязанова Н.Ю.

Москва.
2020 г.

Задание 1

• 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. При демонстрации работы программного комплекса необходимо запустить несколько клиентов (не меньше 5) и продемонстрировать, что сервер обрабатывает обращения каждого запущенного клиента.

Код программы:

info.h

```
#define MSG_LEN 256
#define SOCKET_NAME "socket.soc"
```

client.c

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include "info.h"

int main()
{
   int sockfd = socket(PF_LOCAL, SOCK_DGRAM, 0);
   if (sockfd < 0) {
      printf("Error in socket(). \n");
      return -1;
   }

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

   char msg[MSG_LEN];
```

```
    sprintf(msg, "Message from %d process client", getpid());
    sendto(sockfd, msg, strlen(msg), 0, &server_addr, sizeof(server_addr));
    close(sockfd);

    return 0;
}
```

server.c

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

#define SOCKET_ERR -1
#define BIND_ERR -2
#define RECV_ERR -3

int sockfd;
int sigint_flag = 0;

void sigint_catch(int signum)
{
    close(sockfd);
    unlink(SOCKET_NAME);
    printf("\nCtrl + C catched; Socket closed\n");
    sigint_flag = 1;
}

int main()
{
```

```
sockfd = socket(PF_LOCAL, SOCK_DGRAM, 0);
if (sockfd < 0) {
    printf("Error in socket(). \n");
    return SOCKET_ERR;
}

struct sockaddr client_addr;
client_addr.sa_family = PF_LOCAL;
strcpy(client_addr.sa_data, SOCKET_NAME);

if (bind(sockfd, &client_addr, sizeof(client_addr)) < 0) {
    close(sockfd);
    unlink(SOCKET_NAME);
    printf("Error in bind(). \n");
    return BIND_ERR;
}

printf("Server is running.");
signal(SIGINT, sigint_catch);

char msg[MSG_LEN];
for(;;) {
    int msg_len = recv(sockfd, msg, sizeof(msg), 0);

    if (sigint_flag == 1) {
        return 0;
    }

    if (msg_len < 0) {
        close(sockfd);
        unlink(SOCKET_NAME);
```

```
            printf("Error in recv(). \n");
            return RECV_ERR;
        }

        msg[msg_len] = 0;
        printf("\nMessage from client: %s", msg);
    }

    close(sockfd);
    unlink(SOCKET_NAME);
    printf("Socket closed.");
    return 0;
}
```

Демонстрация работы программы:

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_06/unix$ ./server.out
Server is running.
Message from client: Message from 4118 process client
Message from client: Message from 4119 process client
Message from client: Message from 4120 process client
Message from client: Message from 4121 process client
Message from client: Message from 4122 process client
Message from client: Message from 4123 process client
^CMessage from client: Message from 4124 process client
Ctrl + C catched; Socket closed
```

При этом создаётся сокет (в данной программе с именем socket.soc):

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_06/unix$ ll
итого 64
drwxr-xr-x 2 oleg oleg  4096 апр 21 20:00 ./
drwxr-xr-x 4 oleg oleg  4096 апр 21 19:51 ../
-rw-r--r-- 1 oleg oleg   591 апр 21 18:57 client.c
-rwxr-xr-x 1 oleg oleg 20248 апр 21 19:41 client.out*
-rw-r--r-- 1 oleg oleg    53 апр 21 16:45 info.h
-rw-r--r-- 1 oleg oleg  1443 апр 21 19:04 server.c
-rwxr-xr-x 1 oleg oleg 20752 апр 21 18:56 server.out*
srwxr-xr-x 1 oleg oleg     0 апр 21 20:00 socket.soc=
```

Задание 2

• 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. При демонстрации работы программного комплекса необходимо запустить несколько клиентов (не меньше 5) и продемонстрировать, что сервер обрабатывает обращения каждого запущенного клиента.

Код программы:

info.h

```
#define MSG_LEN 256
#define SOCKET_NAME "socket.soc"
#define SOCK_PORT 8889
#define SOCK_ADDR "localhost"
```

client.c

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

#include "info.h"

#define GETHOSTBYNAME_ERR -1
#define CONNECT_ERR -2
#define SEND_ERR -3
```

```
#define NUMBER_OF_MESSAGES 5

int main(void)
{
    srand(time(NULL));

    int sockfd = socket(PF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        printf("Error in socket().\n");
        return sockfd;
    }

    struct hostent* host = gethostbyname(SOCK_ADDR);
    if (!host) {
        perror("Error in gethostbyname(). \n");
        return GETHOSTBYNAME_ERR;
    }

    struct sockaddr_in server_addr;
    server_addr.sin_family = PF_INET;
    server_addr.sin_port = htons(SOCK_PORT);
    server_addr.sin_addr = *((struct in_addr*) host->h_addr_list[0]);

    if (connect(sockfd, (struct sockaddr*) &server_addr, sizeof(server_addr)) < 0) {
        printf("Error in connect(). \n");
        return CONNECT_ERR;
    }

    char msg[MSG_LEN];
    for (int i = 0; i < NUMBER_OF_MESSAGES; i++) {
        memset(msg, 0, MSG_LEN);
```

```
        sprintf(msg, "%d message.", i);
        printf("%s", msg);

        if (send(sockfd, msg, strlen(msg), 0) < 0) {
            printf("Error in send(). \n");
            return SEND_ERR;
        }

        printf("Sended %d message\n", i);

        sleep(1+  rand() % 3);
    }

    printf("Client ended.\n");
    return 0;
}
```

server.c

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

#define MAX_CLIENTS 10
int clients[MAX_CLIENTS] = { 0 };
#define SOCK_ERR -1
```

```
#define BIND_ERR -2
#define LISTEN_ERR -3
#define ACCEPT_ERR -4
#define SELECT_ERR -5
#define MAX_QUEUE_SIZE 3

int register_new_client(unsigned int fd)
{
    struct sockaddr_in client_addr;
    int addr_size = sizeof(client_addr);

    int new_sock = accept(fd, (struct sockaddr*)&client_addr,
(socklen_t*)&addr_size);
    if (new_sock < 0) {
        printf("Error in accept.");
        return ACCEPT_ERR;
    }

    printf("\nNew client: \nfd = %d \nip = %s:%d\n", new_sock,
        inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));

    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i] == 0) {
            clients[i] = new_sock;
            printf("Registered as client #%d\n", i);
            return 0;
        }
    }
}

void check_client(unsigned int fd, unsigned int client_id)
{
```

```
    char msg[MSG_LEN];
    memset(msg, 0, MSG_LEN);

    struct sockaddr_in client_addr;
    int addr_size = sizeof(client_addr);

    int recv_size = recv(fd, msg, MSG_LEN, 0);
    if (recv_size != 0) {
        msg[recv_size] = '\0';
        printf("Message from %d client: %s\n", client_id, msg);
    }
    else {
        getpeername(fd, (struct sockaddr*)&client_addr, (socklen_t*)&addr_size);
        printf("Client %d disconnected %s:%d \n", client_id,
            inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
        close(fd);
        clients[client_id] = 0;
    }
}

int main(void)
{
    int sockfd = socket(PF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        printf("Error in socket(). \n");
        return SOCK_ERR;
    }

    struct sockaddr_in client_addr;
    client_addr.sin_family = PF_INET;
    client_addr.sin_port = htons(SOCK_PORT);
```

```
client_addr.sin_addr.s_addr = INADDR_ANY;

if (bind(sockfd, (struct sockaddr*) &client_addr, sizeof(client_addr)) < 0) {
    printf("Error in bind.\n");
    return BIND_ERR;
}

printf("Server is running on the %d port.\n", SOCK_PORT);

if (listen(sockfd, MAX_QUEUE_SIZE) < 0) {
    printf("Error in listen.\n");
    return LISTEN_ERR;
}

for (;;) {
    fd_set readfds;
    int max_fd;
    int active_clients_count;

    FD_ZERO(&readfds);
    FD_SET(sockfd, &readfds);
    max_fd = sockfd;

    for (int i = 0; i < MAX_CLIENTS; i++) {
        int fd = clients[i];

        if (fd > 0)
            FD_SET(fd, &readfds);

        max_fd = (fd > max_fd) ? (fd) : (max_fd);
    }
```

```
        active_clients_count = select(max_fd + 1, &readfds, NULL, NULL, NULL);

        if (active_clients_count < 0 && (errno != EINTR)) {
            printf("Error in select. \n");
            return SELECT_ERR;
        }

        if (FD_ISSET(sockfd, &readfds))
            register_new_client(sockfd);

        for (int i = 0; i < MAX_CLIENTS; i++) {
            int fd = clients[i];
            if ((fd > 0) && FD_ISSET(fd, &readfds))
                check_client(fd, i);
        }
    }

    return 0;
}
```

Демонстрация работы программы:

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_06/net$ ./server.out
Server is running on the 8889 port.

New client:
fd = 4
ip = 127.0.0.1:43336
Registered as client #0
Message from 0 client: 0 message.

New client:
fd = 5
ip = 127.0.0.1:43338
Registered as client #1
Message from 1 client: 0 message.
Message from 1 client: 1 message.

New client:
fd = 6
ip = 127.0.0.1:43340
Registered as client #2
Message from 2 client: 0 message.
Message from 0 client: 1 message.
Message from 1 client: 2 message.
Message from 2 client: 1 message.

New client:
fd = 7
ip = 127.0.0.1:43342
Registered as client #3
Message from 3 client: 0 message.
Message from 0 client: 2 message.
Message from 1 client: 3 message.
```

```
New client:
fd = 8
ip = 127.0.0.1:43344
Registered as client #4
Message from 4 client: 0 message.
Message from 2 client: 2 message.
Message from 3 client: 1 message.
Message from 1 client: 4 message.
Message from 2 client: 3 message.
Message from 4 client: 1 message.
Message from 0 client: 3 message.
Message from 3 client: 2 message.
Client 1 disconnected 127.0.0.1:43338

New client:
fd = 5
ip = 127.0.0.1:43346
Registered as client #1
Message from 1 client: 0 message.
Message from 2 client: 4 message.
Message from 4 client: 2 message.
Message from 0 client: 4 message.
Message from 4 client: 3 message.
Message from 3 client: 3 message.
Message from 1 client: 1 message.
Client 0 disconnected 127.0.0.1:43336
Client 2 disconnected 127.0.0.1:43340
Message from 4 client: 4 message.
Message from 3 client: 4 message.
Client 3 disconnected 127.0.0.1:43342
Message from 1 client: 2 message.
Message from 1 client: 3 message.
Client 4 disconnected 127.0.0.1:43344
Message from 1 client: 4 message.
Client 1 disconnected 127.0.0.1:43346
```

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_06/net$ lsof -i :8889
COMMAND     PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
server.ou 3801 oleg    3u  IPv4  58739      0t0  TCP *:8889 (LISTEN)
server.ou 3801 oleg    4u  IPv4  61505      0t0  TCP localhost:8889->localhost:43432 (ESTABLISHED)
server.ou 3801 oleg    5u  IPv4  61520      0t0  TCP localhost:8889->localhost:43454 (ESTABLISHED)
server.ou 3801 oleg    6u  IPv4  59354      0t0  TCP localhost:8889->localhost:43456 (ESTABLISHED)
client.ou 4180 oleg    3u  IPv4  59935      0t0  TCP localhost:43432->localhost:8889 (ESTABLISHED)
client.ou 4190 oleg    3u  IPv4  65408      0t0  TCP localhost:43454->localhost:8889 (ESTABLISHED)
client.ou 4191 oleg    3u  IPv4  59942      0t0  TCP localhost:43456->localhost:8889 (ESTABLISHED)
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_06/net$ lsof -i :8889
COMMAND     PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
server.ou 3801 oleg    3u  IPv4  58739      0t0  TCP *:8889 (LISTEN)
```