Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Институт «Электронных и информационных систем»

Кафедра «Информационных систем и технологий»

**Семейство протоколов TCP/IP. Сокеты в UNIX и работа с ними**

Лабораторная работа №11 по учебной дисциплине «Операционные системы»

По направлению 09.03.01 «Информатика и вычислительная техника»

Отчёт

Принял преподаватель:

\_\_\_\_\_\_\_\_\_ Ананьев В. В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Выполнил студент группы 8091:

\_\_\_\_\_\_\_ Кудряшов И. С.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Великий Новгород

2020

**Цель работы:** познакомиться с механизмами работы сокетов в UNIX.

**Задание:**

Написать программу-клиент и программу-сервер для выполнения следующего сценария: клиент генерирует случайным образом массив целых чисел с заданным размером и диапазоном значений; клиент передаёт по сети массив на сервер; сервер сортирует массив по убыванию и возвращает его клиенту; клиент выводит на экран отсортированный массив и время ожидания ответа от сервера.

Протокол транспортного уровня: UDP.

При запуске программы-сервера через параметры командной строки передаётся порт, на котором будет работать сервер. Если данный порт занят, программа-сервер должна получить порт по усмотрению системы. При успешном запуске сервер должен выдать на экран номер порта, на котором он работает.

При запуске программы-клиента через параметры командной строки передаётся IP-адрес и порт (сокет) программы-сервера в формате <IP-address>:<port> (например, 192.168.0.100:4242). После запуска программы-клиента пользователь вводит размер массива, минимальное и максимальное значение элементов.

Если размер данных больше 508 байт(размер, который по мнению википедии будет принят любым хостом без фрагментации), то данные разбиваются на несколько пакетов. В начале каждого пакета его номер, данные записываются принимающей стороной согласно номеру пакета. Номер добавлен так как udp не гарантирует порядок прихода пакетов.

**Содержание файла udp\_serv.c:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <malloc.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

static int comp(const void \*a, const void \*b)

{

    return \*(const int \*)a - \*(const int \*)b;

}

#define MAX\_SAVE\_UDP\_DATA\_SIZE 508

// Driver code

int main(int argv, char \*argc[])

{

    int MAS\_SIZE;

    if (argv != 2)

    {

        perror("No port given");

        return -1;

    }

    int port = atoi(argc[1]);

    if (port == 0)

    {

        perror("Not correct port");

        return -1;

    }

    int sockfd;

    int \*buffer;

    struct sockaddr\_in servaddr, cliaddr;

    // Creating socket file descriptor

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

    {

        perror("socket creation failed");

        exit(EXIT\_FAILURE);

    }

    memset(&servaddr, 0, sizeof(servaddr));

    memset(&cliaddr, 0, sizeof(cliaddr));

    // Filling server information

    servaddr.sin\_family = AF\_INET; // IPv4

    servaddr.sin\_addr.s\_addr = INADDR\_ANY;

    servaddr.sin\_port = htons(port);

    // Bind the socket with the server address

    if (bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

    {

        servaddr.sin\_port = 0;

        if (bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

        {

            perror("bind failed");

            exit(EXIT\_FAILURE);

        }

    }

    socklen\_t servlen = sizeof(servaddr);

    getsockname(sockfd, (struct sockaddr \*)&servaddr, &servlen);

    printf("Lisening on port: %d\n", ntohs(servaddr.sin\_port));

    int len;

    len = sizeof(cliaddr); //len is value/resuslt

    recvfrom(sockfd, (int \*)&MAS\_SIZE, sizeof(MAS\_SIZE),

             0, (struct sockaddr \*)&cliaddr,

             &len);

    sendto(sockfd, (int \*)&MAS\_SIZE, sizeof(MAS\_SIZE),

           0, (const struct sockaddr \*)&cliaddr,

           len);

    buffer = calloc(MAS\_SIZE, sizeof(int));

    if (MAS\_SIZE \* sizeof(int) <= MAX\_SAVE\_UDP\_DATA\_SIZE)

    {

        recvfrom(sockfd, (int \*)buffer, sizeof(buffer[0]) \* MAS\_SIZE,

                 0, (struct sockaddr \*)&cliaddr,

                 &len);

        qsort(buffer, MAS\_SIZE, sizeof(buffer[0]), comp);

        sendto(sockfd, (int \*)buffer, sizeof(buffer[0]) \* MAS\_SIZE,

               0, (const struct sockaddr \*)&cliaddr,

               len);

    }

    else

    {

        int data\_item\_count = (MAX\_SAVE\_UDP\_DATA\_SIZE - sizeof(int)) / sizeof(int);

        int \*recive\_buf = malloc(MAX\_SAVE\_UDP\_DATA\_SIZE);

        for (int i = 0; i < (MAS\_SIZE / data\_item\_count + 1); i++)

        {

            recvfrom(sockfd, (int \*)recive\_buf, MAX\_SAVE\_UDP\_DATA\_SIZE,

                     0, (struct sockaddr \*)&cliaddr,

                     &len);

            int block\_number = recive\_buf[0];

            int block\_size = block\_number == MAS\_SIZE / data\_item\_count ? MAS\_SIZE % data\_item\_count : data\_item\_count;

            for (int j = 1; j <= block\_size; j++)

                buffer[block\_number \* data\_item\_count + j - 1] = recive\_buf[j];

        }

        free(recive\_buf);

        qsort(buffer, MAS\_SIZE, sizeof(buffer[0]), comp);

        for (int i = 0; i < (MAS\_SIZE / data\_item\_count + 1); i++)

        {

            int block\_size = i == MAS\_SIZE / data\_item\_count ? MAS\_SIZE % data\_item\_count : data\_item\_count;

            int data\_size = sizeof(int) \* (1 + block\_size);

            int \*send\_buf = malloc(data\_size);

            send\_buf[0] = i;

            for (int j = 1; j <= block\_size; j++)

                send\_buf[j] = buffer[i \* data\_item\_count + j - 1];

            sendto(sockfd, (int \*)send\_buf, data\_size,

                   0, (const struct sockaddr \*)&cliaddr,

                   len);

            free(send\_buf);

        }

    }

    free(buffer);

    printf("Server off\n");

    return 0;

}

**Содержание файла udp\_cilent.c:**

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <sys/time.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

int get\_random\_number(int min, int max)

{

    return rand() % (max - min + 1) + min;

}

unsigned long long mic\_s\_time()

{

    struct timeval t;

    gettimeofday(&t, NULL);

    unsigned long long mt = (unsigned long long)t.tv\_sec \* 1000000 + t.tv\_usec;

    return mt;

}

#define MAX\_SAVE\_UDP\_DATA\_SIZE 508

// Driver code

int main(int argv, char \*argc[])

{

    int MAS\_SIZE = 10, MIN\_R = 0, MAX\_R = 1000;

    if (argv != 2)

    {

        perror("No address given");

        return -1;

    }

    int border = strstr(argc[1], ":") - argc[1];

    if (border + argc[1] == NULL)

    {

        perror("Not correct format of address");

        return -1;

    }

    //Set of for make

    printf("Enter count of elements: ");

    scanf("%d", &MAS\_SIZE);

    printf("\nEnter minimum value: ");

    scanf("%d", &MIN\_R);

    printf("\nEnter maximum value: ");

    scanf("%d", &MAX\_R);

    printf("\n");

    char ip[border + 1];

    strncpy(ip, argc[1], border);

    ip[border] = '\0';

    int port = atoi(argc[1] + border + 1);

    int sockfd;

    int \*buffer;

    struct sockaddr\_in servaddr;

    // Creating socket file descriptor

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

    {

        perror("socket creation failed");

        exit(EXIT\_FAILURE);

    }

    memset(&servaddr, 0, sizeof(servaddr));

    // Filling server information

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_port = htons(port);

    inet\_aton(ip, &(servaddr.sin\_addr));

    int len;

    unsigned long long t;

    sendto(sockfd, (int \*)&MAS\_SIZE, sizeof(MAS\_SIZE),

           0, (const struct sockaddr \*)&servaddr,

           sizeof(servaddr));

    int temp;

    recvfrom(sockfd, (int \*)&temp, sizeof(temp),

             0, (struct sockaddr \*)&servaddr,

             &len);

    if (temp != MAS\_SIZE)

    {

        perror("Size sending error");

        return -1;

    }

    buffer = calloc(MAS\_SIZE, sizeof(int));

    srand(time(NULL));

    for (int i = 0; i < MAS\_SIZE; i++)

        buffer[i] = get\_random\_number(MIN\_R, MAX\_R);

    if (MAS\_SIZE \* sizeof(int) <= MAX\_SAVE\_UDP\_DATA\_SIZE)

    {

        sendto(sockfd, (int \*)buffer, sizeof(buffer[0]) \* MAS\_SIZE,

               0, (const struct sockaddr \*)&servaddr,

               sizeof(servaddr));

        t = mic\_s\_time();

        recvfrom(sockfd, (int \*)buffer, sizeof(buffer[0]) \* MAS\_SIZE,

                 0, (struct sockaddr \*)&servaddr,

                 &len);

        t = mic\_s\_time() - t;

    }

    else

    {

        int data\_item\_count = (MAX\_SAVE\_UDP\_DATA\_SIZE - sizeof(int)) / sizeof(int);

        for (int i = 0; i < (MAS\_SIZE / data\_item\_count + 1); i++)

        {

            int block\_size = i == MAS\_SIZE / data\_item\_count ? MAS\_SIZE % data\_item\_count : data\_item\_count;

            int data\_size = sizeof(int) \* (1 + block\_size);

            int \*send\_buf = malloc(data\_size);

            send\_buf[0] = i;

            for (int j = 1; j <= block\_size; j++)

                send\_buf[j] = buffer[i \* data\_item\_count + j - 1];

            sendto(sockfd, (int \*)send\_buf, data\_size,

                   0, (const struct sockaddr \*)&servaddr,

                   sizeof(servaddr));

            free(send\_buf);

        }

        t = mic\_s\_time();

        int \*recive\_buf = malloc(MAX\_SAVE\_UDP\_DATA\_SIZE);

        for (int i = 0; i < (MAS\_SIZE / data\_item\_count + 1); i++)

        {

            recvfrom(sockfd, (int \*)recive\_buf, MAX\_SAVE\_UDP\_DATA\_SIZE,

                     0, (struct sockaddr \*)&servaddr,

                     &len);

            int block\_number = recive\_buf[0];

            int block\_size = block\_number == MAS\_SIZE / data\_item\_count ? MAS\_SIZE % data\_item\_count : data\_item\_count;

            for (int j = 1; j <= block\_size; j++)

                buffer[block\_number \* data\_item\_count + j - 1] = recive\_buf[j];

        }

        free(recive\_buf);

        t = mic\_s\_time() - t;

    }

    printf("Server : ");

    for (int i = 0; i < MAS\_SIZE; i++)

        printf("%d ", buffer[i]);

    printf("\n");

    printf("It took %lld microseconds.\n", t);

    close(sockfd);

    free(buffer);

    return 0;

}

**Вывод:** В процессе выполнения лабораторной работы я на практике познакомился с работой сокетов в UNIX.