

**Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования**

**НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО**



УНИВЕРСИТЕТ ИТМО

Факультет безопасности информационных технологий
Дисциплина: «Системное программирование»

ОТЧЕТ

Лабораторная работа № 2

«Процессы, потоки и средства межпроцессного взаимодействия»

Вариант 24

Выполнил студент группы N3351: Ярьсько С.А. _____

Проверил: Гирик А.В. _____

Дата: _____

Санкт-Петербург, 2020

Цель работы:

Написать на языке C две программы для POSIX-совместимой ОС: совместимой ОС:

- сервер, поддерживающий заданный вариантом тип многозадачности (Табл. 2), транспортный протокол (Табл. 3) и прикладной протокол (Табл. 4);
- клиент, поддерживающий заданный вариантом протокол и предназначенный для тестирования сервера.

Вариант 24. Задание:

Тип многозадачности: Многопроцессность (создание рабочих задач с помощью вызова `fork`).

Транспортный протокол: TCP

Запрос: строка, содержащая выражение вида `OP X1 ... XN`, завершающаяся символом LF, где `OP` — одна из поразрядных операций `AND`, `OR`, `XOR`, `NAND` или `NOR`, `X1 ... XN` — целочисленные операнды, записанные в десятичной или шестнадцатеричной форме, разделенные пробельными символами.

Ответ, если ошибок не было: строка, содержащая результат операции из запроса, завершающаяся символом LF.

Ответ, если были ошибки: строка «`ERROR N`», завершающаяся символом LF, где `N` — код ошибки.

Клиент:

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

// Проверка выполнения функций на ошибки
void CHECK_RESULT(int res, char * msg)
{
    do {
        if (res < 0) {
            perror(msg);
            exit(EXIT_FAILURE); }
    } while (0);
}

#define BUF_SIZE 1024

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

    // Для создание соединения
    int clientSocket;
    char buffer[BUF_SIZE] = {0};
    struct sockaddr_in serverAddr = {0};

    // Для getopt()
    int option = 0;
    unsigned short int opt_port = 0;
    char * opt_ip = NULL;

    // Для getline()
    char * str = NULL;
    char * str_copy = NULL;
    size_t len = 0;

    // Переменные окружения
    setenv("L2PORT", "5555", 1);
    setenv("L2ADDR", "127.0.0.1", 1);

    // Принятие ключей через getopt()
    while ( (option = getopt(argc,argv,"a:p:vh")) != -1)
    {
        switch (option)
        {
            {
                case 'a': opt_ip = optarg;
break;
                case 'p': opt_port = (unsigned short int)atoi(optarg);
break;
                case 'v': printf("Current version: 1.0.1\n");
return 0;
                case 'h': printf("lab2client -a \"IP\" | -p \"PORT\" | -v \"is version\"\n");
return 0;
                case '?': printf("Invalid argument\n");
return -1;
            }
        }
    }
```

```

// Замена переменных на переменные окружения, если не переданы через строку
if (opt_ip==NULL) opt_ip = getenv("L2ADDR");
if (opt_port==0) opt_port = (unsigned short int)atoi(getenv("L2PORT"));

while (getline(&str, &len, stdin))
{
    // Создание сокета
    clientSocket = socket(AF_INET, SOCK_STREAM, 6);
    CHECK_RESULT(clientSocket, "socket");

    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = inet_addr(opt_ip);
    serverAddr.sin_port = htons(opt_port);

    // Установление соединения с сокетом
    int res = connect(clientSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr));
    CHECK_RESULT(res, "connect");

    // Отправка сообщения в сокет
    res = send(clientSocket, str, strlen(str)+1, 0);
    CHECK_RESULT(res, "sendto");

    // Закрытие соединения на отправку серверу
    res = shutdown(clientSocket, SHUT_WR);
    CHECK_RESULT(res, "shutdown");

    // Получение сообщения от сокета
    res = recv(clientSocket, buffer, BUF_SIZE, 0);
    CHECK_RESULT(res, "recvfrom");
    printf("%s\n", buffer);

    // Закрытие сокета
    close(clientSocket);
}

return 0;
}

```

Сервер:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <sys/resource.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <errno.h>
#include <signal.h>

// Перевод строки с dec или hex в long int
long int to_dec(char * num)
{
    long int res_to_dec = 0;
    if ( num[0]=='0' && num[1]=='x' )
        res_to_dec = strtol(num, 0, 16);
    else res_to_dec = atoi(num);
    return res_to_dec;
}

// Расчет результата запроса
long int calc(char * str)
{
    char * oper = NULL;
    char * num1 = NULL;
    char * num2 = NULL;
    long int res = 0;

    oper = strtok(str, " \n\r\t\v");
    num1 = strtok(NULL, " \n\r\t\v");
    num2 = strtok(NULL, " \n\r\t\v");
    int oper_num = oper[0] + oper[1] + oper[2];

    switch (oper_num)
    {
        case 211: res = to_dec(num1) & to_dec(num2); break; //AND
        case 161: res = to_dec(num1) | to_dec(num2); break; //OR
        case 249: res = to_dec(num1) ^ to_dec(num2); break; //XOR
        case 221: res = ~to_dec(num1) | ~to_dec(num2); break; //NAND
        case 239: res = ~to_dec(num1) & ~to_dec(num2); break; //NOR
    }

    num2 = strtok(NULL, " \n\r\t\v");

    while (num2)
    {
        switch (oper_num)
        {
            case 211: res = res & to_dec(num2); break; //AND
            case 161: res = res | to_dec(num2); break; //OR
            case 249: res = res ^ to_dec(num2); break; //XOR
            case 221: res = ~res | ~to_dec(num2); break; //NAND
            case 239: res = ~res & ~to_dec(num2); break; //NOR
        }
        num2 = strtok(NULL, " \n\r\t\v");
    }
    return res;
}
```

```

// Проверка на правильность введенного запроса
int check_str(char * str)
{
    char * num = NULL;
    int k = 1;

    if ( !( !strcmp(str, "AND ", 4) || !strcmp(str, "OR ", 3) || !strcmp(str, "XOR ", 4)
        || !strcmp(str, "NAND ", 5) || !strcmp(str, "NOR ", 4) ) )
        return -1;

    strtok(str, " \n\r\t\v");
    num = strtok(NULL, " \n\r\t\v");

    while (num)
    {
        if ( num[0]=='0' && num[1]=='x' )
        {
            for (int i=2; i < (int)strlen(num); i++)
                if (strchr("0123456789ABCDEF", num[i])==NULL)
                    return -16;
        }
        else
        {
            for (int i=0; i < (int)strlen(num); i++)
                if (strchr("-0123456789", num[i])==NULL)
                    return -10;
        }
        num = strtok(NULL, " \n\r\t\v");
        k++;
    }
    if (k<3) return 3;
    return 0;
}

// Возврат текущего времени в нужной форме
char * settime()
{
    char s[40];
    char *tmp;
    struct tm *u;
    const time_t timer = time(NULL);
    u = localtime(&timer);
    for (int i = 0; i<40; i++) s[i] = 0;
    strftime(s, 40, "\n%d.%m.%Y %H:%M:%S\t", u);
    tmp = (char*)malloc(sizeof(s));
    strcpy(tmp, s);
    return(tmp);
}

// Проверка выполнения функций на ошибки
void CHECK_RESULT(int res, char * msg)
{
    extern FILE * logfile;
    do {
        if (res < 0) {
            printf("ERROR %s %d", msg, errno);
            fprintf(logfile, "%sERROR %s %d",settime(), msg, errno);
            exit(EXIT_FAILURE); }
    } while (0);
}

```

```

// Обработчик запросов
void action_handler(int sig)
{
    extern volatile unsigned short int sig_flag;
    // int res;
    switch (sig)
    {
        // case SIGCHLD:
        //     do { res=waitpid(-1, 0, WNOHANG); }
        //     while ( res != -1);
        //     break;
        case SIGUSR1:
            sig_flag = 1;
            break;
        case SIGINT:
        case SIGTERM:
        case SIGQUIT:
            sig_flag = 2;
    }
}

// Действия при SIGUSR1
void sig_usr1()
{
    extern int request_error, request_total;
    extern FILE * logfile;
    extern time_t start, end;
    extern volatile unsigned short int sig_flag;
    time(&end);
    fprintf(logfile, "%sSIGUSR1 Time: %ld sec, Succeeded: %d, Failed: %d", settime(),
        end-start, request_total-request_error, request_error);
    fprintf(stderr, "%sSIGUSR1 Time: %ld sec, Succeeded: %d, Failed: %d\n", settime(),
        end-start, request_total-request_error, request_error);
    fflush(logfile);
    fflush(stderr);
    sig_flag = 0;
}

// Действия при SIGINT, SIGTERM, SIGQUIT
void sig_int_term_quit()
{
    extern int serverSocket, clientSocket;
    extern FILE * logfile;
    fprintf(logfile, "%sSIGINT", settime());
    close(clientSocket);
    fprintf(logfile, "%sclose %d", settime(), clientSocket);
    close(serverSocket);
    fprintf(logfile, "%sclose %d", settime(), serverSocket);
    fprintf(logfile, "%sfclose logfile", settime());
    fflush(logfile);
    fclose(logfile);
    exit(0);
}

```

```

int do_main()
{
    extern int serverSocket, clientSocket;
    extern int request_error, request_total;
    extern FILE * logfile;
    extern time_t start, end;
    extern volatile unsigned short int sig_flag;

    extern unsigned short int opt_daemon;
    extern unsigned short int opt_port;
    extern char * opt_ip;
    extern unsigned short int opt_sleep;
    extern char * opt_logfile;

    extern struct sigaction sa;
    extern sigset_t set;

    time(&start);
    int res=0, status=0, check_str_res=0;
    int file_pipes[2];
    long int calc_res = 0;
    pid_t pid_fork;

    // Для передачи сообщений
    char buffer[1024];
    char * buffer_tmp_calc = NULL;
    char * buffer_tmp_check = NULL;

    // Для создание соединения
    struct sockaddr_in serverAddr = {0};
    struct sockaddr_storage serverStorage;
    socklen_t addr_size;

    // Заполнение набора сигналов и назначение обработчика сигналов
    sigemptyset(&set);
    sigaddset(&set, SIGINT );
    sigaddset(&set, SIGTERM );
    sigaddset(&set, SIGQUIT);
    sigaddset(&set, SIGUSR1 );
    // sigaddset(&set, SIGCHLD );
    sa.sa_handler = action_handler;

    // Установка обработчика сигналов на сигналы
    res = sigaction(SIGINT, &sa, NULL);
    CHECK_RESULT(res, "sigaction SIGINT");
    res = sigaction(SIGTERM, &sa, NULL);
    CHECK_RESULT(res, "sigaction SIGTERM");
    res = sigaction(SIGQUIT, &sa, NULL);
    CHECK_RESULT(res, "sigaction SIGQUIT");
    res = sigaction(SIGUSR1, &sa, NULL);
    CHECK_RESULT(res, "sigaction SIGUSR1");
    // res = sigaction(SIGCHLD, &sa, NULL);
    // CHECK_RESULT(res, "sigaction SIGCHLD");
    sigset(SIGCHLD, SIG_IGN);

    // Создание ЛОГ-файла
    logfile = fopen(opt_logfile, "a");
    if (logfile == NULL) {
        printf("Error accessing log file\n");
        return -1; }
    if(opt_daemon)
        fprintf(logfile, "%sdaemon %d", settime(), getpid());
    fprintf(logfile, "%sfopen %s", settime(), opt_logfile);

    // Создание сокета
    serverSocket = socket(AF_INET, SOCK_STREAM, 6);
    CHECK_RESULT(serverSocket, "socket");
    fprintf(logfile, "%ssocket %d", settime(), serverSocket);

```



```

// Установка параметров протокола и уровня сокета
res = setsockopt(serverSocket, SOL_SOCKET, SO_REUSEADDR, &(int){ 1 }, sizeof(int));
CHECK_RESULT(res, "setsockopt");
fprintf(logfile, "%ssetsockopt %d", settime(), serverSocket);

serverAddr.sin_family = AF_INET;
serverAddr.sin_addr.s_addr = inet_addr(opt_ip);
serverAddr.sin_port = htons(opt_port);

// Привязка сокета к локальному адресу
res = bind(serverSocket, (struct sockaddr *)&serverAddr, sizeof(serverAddr));
CHECK_RESULT(res, "bind");
fprintf(logfile, "%sbind %d", settime(), serverSocket);

// Сообщение сокету, что должны приниматься новые соединения
res = listen(serverSocket, 5);
CHECK_RESULT(res, "listen");
fprintf(logfile, "%slisten %d", settime(), serverSocket);

fflush(logfile);

while (1)
{
    sigprocmask(SIG_BLOCK, &set, NULL);

    // Получение нового сокета для нового входящего соединения от клиента
    addr_size = sizeof(serverStorage);
    clientSocket = accept(serverSocket, (struct sockaddr *)&serverStorage, &addr_size);
    CHECK_RESULT(clientSocket, "accept");
    fprintf(logfile, "%saccept %d", settime(), clientSocket);
    fflush(logfile);

    sigprocmask(SIG_UNBLOCK, &set, NULL);

    // Проверка флага получения сигнала
    if (sig_flag==2) sig_int_term_quit();
    else if (sig_flag==1) sig_usr1();

    // Создание канала для общения процессов
    res = pipe(file_pipes);
    CHECK_RESULT(res, "pipe");
    fprintf(logfile, "%spipe", settime());

    // Начало дочернего процесса
    pid_fork = fork();
    if (pid_fork==0)
    {
        close(file_pipes[0]);

        // Получение сообщения от сокета
        res = recv(clientSocket, buffer, 1024, 0);
        CHECK_RESULT(res, "recv");
        fprintf(logfile, "%srecv %d %s", settime(), clientSocket, buffer);
        fflush(logfile);
        request_total++;
        write(file_pipes[1], &request_total, sizeof(int));

        // Проверка запроса и заполнение буфера результатом обработки запроса
        buffer_tmp_calc = strdup(buffer);
        buffer_tmp_check = strdup(buffer);
        if ( ( check_str_res = check_str(buffer_tmp_check) ) !=0 ) {
            sprintf(buffer, "ERROR %d\n", check_str_res);
            fprintf(logfile, "%scheck_str ERROR %d", settime(), check_str_res);
            fflush(logfile);
            request_error++;
            write(file_pipes[1], &request_error, sizeof(int)); }
    }
}

```

```

else {
    calc_res = calc(buffer_tmp_calc);
    sprintf(buffer, "%ld\n", calc_res); }
close(file_pipes[1]);

// Приостановка обслуживающего запрос процесса
if (opt_sleep!=0) {
    fprintf(logfile, "%ssleep %d", settime(), opt_sleep);
    fflush(logfile);
    sleep(opt_sleep); }

// Отправка сообщения в сокет
res = send(clientSocket, buffer, strlen(buffer)+1, 0);
CHECK_RESULT(res, "send");
fprintf(logfile, "%ssend %d %s", settime(), clientSocket, buffer);

// Заккрытие соединения на отправку клиенту
res = shutdown(clientSocket, SHUT_WR);
fprintf(logfile, "%sshutdown %d", settime(), clientSocket);

// Заккрытие сокета для входящего соединения от клиента
close(clientSocket);
fprintf(logfile, "%sclose %d", settime(), clientSocket);

fflush(logfile);
exit(0);
}

// Читаем из неименованного канала
close(file_pipes[1]);
read(file_pipes[0], &request_total, sizeof(int));
read(file_pipes[0], &request_error, sizeof(int));
close(file_pipes[0]);
fflush(logfile);

// Проверка флага получения сигнала
if (sig_flag==2) sig_int_term_quit();
else if (sig_flag==1) sig_usr1();
}

// Заккрытие сокета сервера
close(serverSocket);
fprintf(logfile, "%sclose %d", settime(), serverSocket);
fclose(logfile);
return 0;
}

// Глобальные переменные
int serverSocket, clientSocket;
int request_error, request_total;
FILE * logfile;
time_t start, end;
volatile unsigned short int sig_flag = 0;

int option = 0;
unsigned short int opt_daemon = 0;
unsigned short int opt_port = 0;
char * opt_ip = NULL;
unsigned short int opt_sleep = 0;
char * opt_logfile = NULL;

struct sigaction sa;
sigset_t set;

```

```

int main(int argc, char *argv[])
{
    // Для демона
    pid_t pid_fork_first, pid_fork_second;
    int daemon_pipes[2], daemon_notify=0;
    FILE * pid_file;

    // Задание переменных окружения
    setenv("L2PORT", "5555", 0);
    setenv("L2ADDR", "127.0.0.1", 0);
    setenv("L2WAIT", "0", 0);
    setenv("L2LOGFILE", "/tmp/lab2.log", 0);

    // Принятие ключей через getopt()
    while ( (option = getopt(argc,argv,"w:dl:a:p:vh")) != -1)
    {
        switch (option)
        {
            case 'w': opt_sleep = (unsigned short int)atoi(optarg); break;
            case 'd': opt_daemon = 1; break;
            case 'l': opt_logfile = optarg; break;
            case 'a': opt_ip = optarg; break;
            case 'p': opt_port = (unsigned short int)atoi(optarg); break;
            case 'v': printf("Current version: 1.0.1\n"); return 0;
            case 'h': printf("lab2server -a \"%IP\" | -p \"%PORT\" | -w \"%SLEEP_TIME\" |  

                -l \"%LOG_FILE\" | \n-d \"is daemon mode\" | -v \"is version\"\\n"); return 0;
            case '?': printf("Invalid argument\n"); return -1;
        }
    }

    // Замена полученных аргументов на переменные окружения, если не переданы через строку
    if (opt_ip==NULL) opt_ip = getenv("L2ADDR");
    if (opt_port==0) opt_port = (unsigned short int)atoi(getenv("L2PORT"));
    if (opt_sleep==0) opt_sleep = (unsigned short int)atoi(getenv("L2WAIT"));
    if (opt_logfile==NULL) opt_logfile = getenv("L2LOGFILE");

```

```

if (opt_daemon==1)
{
    // Никаких файлов не открыто /*1*/
    sa.sa_handler = SIG_DFL; /*2*/
    // sigprocmask до этого не вызывалась /*3*/
    // откатить переменные окружения, которые могут помешать, вроде не должно мешат /*4*/
    pid_fork_first = fork(); /*5*/
    if (pid_fork_first == -1) {
        printf("Error creating Daemon in fork #1\n");
        return -1; }
    else if (pid_fork_first == 0)    // Первый дочерний
    {
        setsid(); /*6*/
        pid_fork_second = fork(); /*7*/
        if (pid_fork_second == -1) {
            printf("Error creating Daemon in fork #2\n");
            return -1; }
        else if (pid_fork_second == 0)    // Второй дочерний == Димон
        {
            close(STDIN_FILENO); /*9*/
            close(STDOUT_FILENO); /*9*/
            close(STDERR_FILENO); /*9*/
            umask(0); /*10*/
            chdir("/"); /*11*/
            pid_file = fopen("/run/lab2server.pid", "w");
            fprintf(pid_file, "%d", getpid()); /*12*/
            fclose(pid_file);
            // сброс привилегий делать не буду, если что setuid() /*13*/
            daemon_notify = 666;
            close(daemon_pipes[0]);
            write(daemon_pipes[1], &daemon_notify, sizeof(int)); /*14*/
            close(daemon_pipes[1]);
            do_main();
        }
        else    // Первый дочерний
            exit(0); /*8*/
    }
    else {    // Main
        close(daemon_pipes[1]);
        read(daemon_pipes[0], &daemon_notify, sizeof(int));
        if (daemon_notify==666) exit(0); } /*15*/
    }
    else do_main();
}

```

Makefile:

```
all:
    gcc -o /usr/local/bin/lab2server Lab2Server.c
    gcc -o /usr/local/bin/lab2client Lab2Client.c
```

Примеры работы программы:

The image shows three terminal windows illustrating the workflow of the Lab2 program.

Top Left Window (SP_Lab2_Server): Shows the compilation of the server program.

```
root@root: ~/Documents/QT Projects/SP_Lab2_Server
root@root:~/Documents/QT Projects/SP_Lab2_Server# gcc -o main main.c
root@root:~/Documents/QT Projects/SP_Lab2_Server# ./main
```

Top Right Window (/tmp): Shows the execution of the server program, displaying network activity logs.

```
root@root: /tmp# tail -f lab2.log
20.12.2019 12:07:44 fopen /tmp/lab2.log
20.12.2019 12:07:44 pipe
20.12.2019 12:07:44 socket 6
20.12.2019 12:07:44 setsockopt 6
20.12.2019 12:07:44 bind 6
20.12.2019 12:07:44 listen 6
20.12.2019 12:07:58 accept 7
20.12.2019 12:08:15 recvfrom 7 OR 24 42 12 65 34 64 23
20.12.2019 12:08:15 sendto 7 127
20.12.2019 12:08:15 shutdown 7
20.12.2019 12:08:15 close 7
20.12.2019 12:08:15 accept 8
20.12.2019 12:08:43 recvfrom 8 XOR 0xFD45 563 -634 0x343F4 -532 244968
20.12.2019 12:08:43 sendto 8 244968
20.12.2019 12:08:43 shutdown 8
20.12.2019 12:08:43 close 8
20.12.2019 12:08:43 accept 9
20.12.2019 12:10:50 recvfrom 9 NOR 532 0x43F 43
20.12.2019 12:10:50 sendto 9 1556
20.12.2019 12:10:50 shutdown 9
20.12.2019 12:10:50 close 9
20.12.2019 12:10:50 accept 10
```

Bottom Left Window (Lab2Files): Shows the compilation of the client program and its execution.

```
root@root:~/Documents/Lab2Files# make
gcc -o /usr/local/bin/lab2server Lab2Server.c
gcc -o /usr/local/bin/lab2client Lab2Client.c
root@root:~/Documents/Lab2Files# lab2server -v
Current version: 1.0.1
^Croot@root:~/Documents/Lab2Files# lab2server
20.12.2019 12:20:46 SIGUSR1 Time: 45, Succeeded: 0, Failed: 0ERROR accept 4
root@root:~/Documents/Lab2Files#
```

Bottom Right Window (/tmp): Shows the execution of the client program, displaying network activity logs.

```
root@root: /tmp# tail -f lab2.log
20.12.2019 12:19:37 fopen /tmp/lab2.log
20.12.2019 12:19:37 pipe
20.12.2019 12:19:37 socket 6
20.12.2019 12:19:37 setsockopt 6
20.12.2019 12:19:37 bind 6
20.12.2019 12:19:37 listen 6
20.12.2019 12:19:53 SIGINT
20.12.2019 12:19:53 close 0
20.12.2019 12:19:53 close 6
20.12.2019 12:19:53 fclose logfile
20.12.2019 12:20:01 fopen /tmp/lab2.log
20.12.2019 12:20:01 pipe
20.12.2019 12:20:01 socket 6
20.12.2019 12:20:01 setsockopt 6
20.12.2019 12:20:01 bind 6
20.12.2019 12:20:01 listen 6
20.12.2019 12:20:46 SIGUSR1 Time: 45, Succeeded: 0, Failed: 0
20.12.2019 12:20:46 ERROR accept 4
```