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»

Лабараторная работа №1

По дисциплине «Операционные системы»

Студент: Барсуков Н.М

Группа: ИУ7-66Б

Преподаватель: Рязанова Н.Ю.

Оценка (баллы): _____

Москва.

2020г.

Листинг кода:

```
#include <syslog.h>
#include <fcntl.h>
#include <sys/resource.h>

#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include <arpa/inet.h>
#include <string.h>
#include <stdarg.h>
#include <errno.h>



const char demonname[64] = "Daemon";
#define LOCKFILE "/var/run/Daemon.pid"

int sock;                       //файловый дескриптор сокета, в который пишем репорты
struct sockaddr_in addr;  //адрес, связанный с этим сокетом

//печать информации путем посылания через сокет
void func_pr(const char* format, ...){
        char message[512];
        va_list ap;
        va_start( ap, format );
        vsnprintf( message, 512, format, ap );
```

```
	va_end( ap );
	//открываем сокет
	sock = socket( AF_INET, SOCK_DGRAM, IPPROTO_UDP );

	sendto( sock, message, strlen(message), 0, (struct sockaddr*) &addr, sizeof(addr) );

	close( sock ); //закрываем сокет

}

int demonize(const char *process_name){
	int i;

	func_pr( "fork" );
	/*выводим демона в отдельный процесс чтобы командная оболочка думала, что команда была выполнена
	 пока не лидер группы позволяет вызвать setsid*/
	int pid = fork();
	if ( pid < 0 ){
		func_pr( "ERROR: fork" );
		return -1;
	}
	if ( pid != 0 ) return 1; //parent

	setsid();
```

```
struct sigaction sa;
sa.sa_handler = SIG_IGN;
sigemptyset( &sa.sa_mask );
sa.sa_flags = 0;

func_pr( "sigaction" );
if ( sigaction(SIGHUP, &sa, NULL) < 0 ){
        func_pr( "ERROR: can't ignore SIGHUP" );
        return -1;
}

func_pr( "changing dir" );

if ( chdir("/") < 0 ){
        func_pr( "ERROR: can't change directory" );
        return -1;
}

func_pr( "umask and getrlimit\n" );
umask(0);


//получаем максимальное количество дескрипторов
struct rlimit rl;
if ( getrlimit(RLIMIT_NOFILE, &rl) < 0 ) {
        func_pr( "ERROR: couldn't get max number of fd" );
        return -1;
}
```

```
        if ( rl.rlim_max == RLIM_INFINITY ) rl.rlim_max = 1024;



        for (i = 0; i < rl.rlim_max; ++i){
                if ( i != sock ) //кроме лог-сокета, чтобы было куда писать
                        close(i);
        }
        func_pr( "\n" );

        int fd0, fd1, fd2;
        fd0 = open( "/dev/null", O_RDWR );
        fd1 = dup(0); //дублируем дескрипторы
        fd2 = dup(0);

        openlog( process_name, LOG_CONS, LOG_DAEMON );
        if ( fd0 != 0 || fd1 != 1 || fd2 != 2){
                func_pr( "ERROR: Incorrect file descriptors 0,1,2" );
                syslog( LOG_ERR, "incorrect fd: %d %d %d", fd0, fd1, fd2 );
                return -1;
        }
        func_pr( "Demonizing finished." );
        return 0;
}

int already_running(void){
#define BUFLEN 8
        int fd;
        char buf[BUFLEN];
```

```
	fd = open( LOCKFILE, O_RDWR|O_CREAT,
S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH );
	if (fd < 0){
		func_pr( "ERROR: unable to open %s: %s", LOCKFILE,
strerror(errno) );
		syslog( LOG_ERR, "unable to open %s: %s", LOCKFILE,
strerror(errno) );
		exit(1);
	}

	flock( fd, LOCK_EX|LOCK_NB );
	if ( errno == EWOULDBLOCK ){
		func_pr( ">>> Can't lock file %s: %s\n>>> Is demon already
launched?", LOCKFILE, strerror(errno) );
		syslog(LOG_ERR, "unable to lock %s: %s", LOCKFILE,
strerror(errno));
		close(fd);
		return 1;
	}

	func_pr( "%d - Successfully locked the file.", getpid() );
	snprintf( buf, BUFLEN, "%d", getpid() );
	write( fd, buf, strlen(buf) );
	return 0;
}

void demonFunc(){
	func_pr( "(%d) timemark...", getpid() );
	syslog( LOG_INFO, "(%d) timemark...\n", getpid() );
	sleep(3);
```

```
}

int main(void)
{
        addr.sin_addr.s_addr = 0x0100007f;
        addr.sin_port = htons(7777);
        addr.sin_family = AF_INET;



        printf( "Demonize...\n" );
        int d = demonize( demonname );
        if ( d > 0 ){
                func_pr( "Parent: finished successfully" );
                return 0;
        }
        else if ( d < 0 ){
                func_pr( "Parent: something went wrong!" );
                return 1;
        }

        if ( already_running() ) return 1;
        func_pr( "Successfully demonized." );

        while ( 1 ) {demonFunc();}
}
```

Программа создает нового демона, проверяет существует ли другой такой демон (already_running). Если существует, то выводитсообщение об этом в syslog и завершает свою работу. Если не существует, то каждые 3 секунды выводит в syslog системное время.

Чтобы создать нового демона нужно:

1) Вызвать umask для сброса маски создания файлов
    a. маска наследуется и может маскировать биты прав доступа (запись, чтение, выполнение)

2) Вызвать fork() и завершить предка
    a. чтобы командная оболочка думала, что команда была выполнена
    b. чтобы новый процесс гарантированно не был лидером группы, что позволит вызвать setsid (у дочернего процесса id отличный от родителя, а pgid наследуется)

3) Создать новую сессию, вызвав setsid, тогда процесс станет:
    a. лидером новой сессии
    b. лидером новой группы процессов
    c. лишится управляющего терминала (TTY = ?)

4) Сделать корневой каталог текущим рабочим каталогом
    a. если рабочий каталог на смонтированной файловой системе, то её нельзя будет отмонтировать, так как процессы-демоны обычно живут, пока система не перезагрузится

5) Закрыть все ненужные открытые файловые дескрипторы, которые процесс-демон может унаследовать и препятствовать их закрытию (для этого нужно сначала получить максимально возможный номер файлового дескриптора (см. код))

6) Такой процесс не связан ни с каким терминальным устройством и не может взаимодействовать с пользователем в интерактивном режиме, даже если он был запущен в рамках интерактивной сессии, он все равно будет переведен в фоновый режим (некоторые процессы-демоны открывают файловые дескрипторы 0 1 и 2 на dev/null - "пустые" stdin, stdout, stderr, что позволяет вызывать в них функции стандартного ввода вывода, не получая при этом ошибок)