figure.1

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВОЙ ПРОЕКТ
ЗАЩИЩЕН С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ

| ст.преп. | | Поляк М.Д. |
|---|---|---|
| должность, уч. степень, звание | подпись, дата | инициалы, фамилия |

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ

## ДЕМОН ДЛЯ ВЕБКАМЕРЫ

по дисциплине: ОПЕРАЦИОННЫЕ СИСТЕМЫ И СЕТИ

РАБОТУ ВЫПОЛНИЛ

| СТУДЕНТ ГР. | 4336 | | Канифатов Д.И. |
|---|---|---|---|
| | | подпись, дата | инициалы, фамилия |

Санкт-Петербург
2016

# 1 Цель работы

Знакомство с устройством ядра ОС Linux. Получение опыта разработки драйвера устройства.

# 2 Цель работы

Добавление защиты от несанкционированного запуска операционной системы. Необходимо внести изменения в процесс загрузки ядра Linux, добавив проверку наличия подключенного через интерфейс USB flash-накопителя с заданным серийным номером. Если в процессе загрузки операционной системы нужный flash-накопитель подключен к одному из портов USB, то операционная система успешно загружается в штатном режиме. Если flash-накопитель с нужным серийным номером отсутствует, отобразить на экране предупреждение и таймер с обратным отсчетом (30 секунд), загрузка операционной системы при этом приостанавливается. По истечении обратного отсчета таймера должно происходить автоматическое выключение компьютера. При подключении к любому из USB-портов нужного flash-накопителя во время обратного отсчета таймера, таймер должен останавливаться, после чего операционная система должна загружаться в штатном режиме.

# 3 Техническая документация

1. Сборка проекта:
   Скачиваем файлы с репозитория на github при помощи команды:

   ```
   git clone https://github.com/dimitrR/WebdemonLinux
   ```

2. Сборка демона для вебкамеры:

   1)Устанавливаем в файле mydemon.conf путь сохранения фотографий, и временной промежуток снятие снимков с помощью демона.

   2)Сборка демона происходит с помощью использования скрипта make.sh:

   rm webdemon
   g++ -c -o settingdemon.o settingdemon.cpp
   g++ -c -o videodevice.o videodevice.cpp
   g++ -c -o main.o main.cpp
   g++ settingdemon.o videodevice.o main.o -o webdemon -ljpeg

   3)Для запуска скрипта make.sh требуется прописать в консоле (./make.sh) или запустить скрипт двойным нажатием левой кнопкой мыши.

4)В корневой папке проекта появляется файл webdemon

5)Запускаем демон нажатием на файл webdemon или же прописываем в терминале (./webdemon)

# 4 Скриншоты

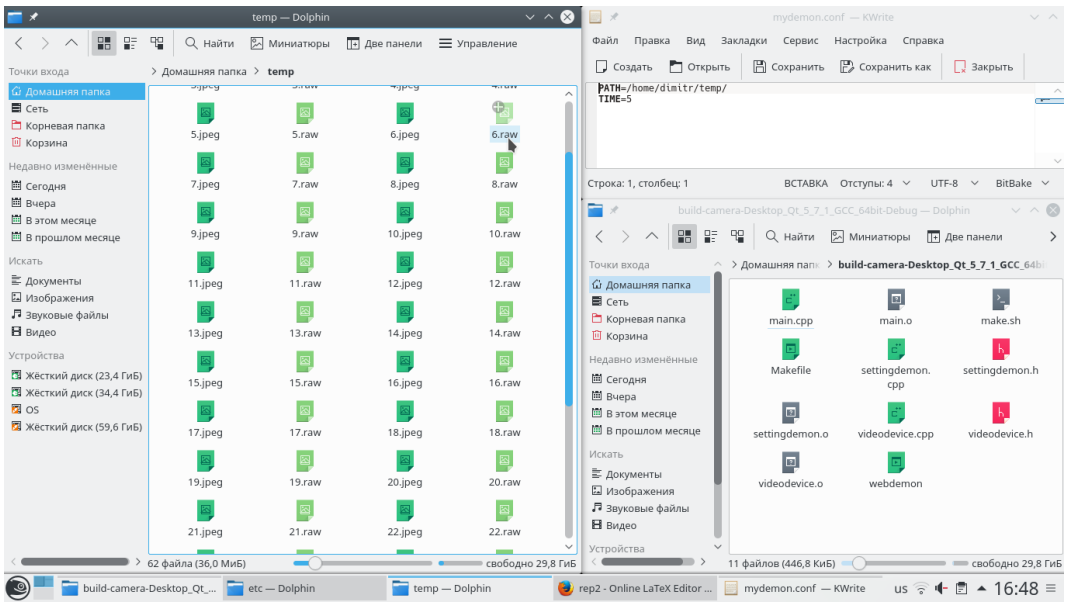


Рис. 1:
Файл настроек и снимки

# 5 Заключение

В процессе выполнения данной курсовой работы мною были получены знания и навыки, необходимые для работы с ядром ОС Linux, а так же знания и навыки в разработке драйверов устройств.

# 6 Приложение

main.cpp

```
#include <errno.h>      //errno
#include <string.h>     //string
#include <stdio.h>      //cout
#include <unistd.h>     //fork();
#include <sys/stat.h>   //umask();

#include "videodevice.h"
#include "settingdemon.h"

void demonBody();

int main(){
    //создаем потомка
    int pid=fork();

    if(pid==-1){ // если не удалось запустить потомка
        cout<<"Error: Start Daemon failed ("<<strerror(errno)<<")"<<endl;/
        return -1;
    }
    else
        if(!pid){
            //потомок
            // разрешаем выставлять все биты прав на создаваемые файлы, ин
            umask(0);

            //создаём новый сеанс, чтобы не зависеть от родителя
            setsid();

            //переходим в корень диска, если мы этого не сделаем
            chdir("/");

            //закрываем дискрипторы ввода/вывода/ошибок, так как нам они б
            close(STDIN_FILENO);
            close(STDOUT_FILENO);
            close(STDERR_FILENO);

            //выполняем основной код демона
            demonBody();

            return 0;
       }
       else{
```

```
            //родитель, завершим процес, т.к. основную свою задачу (запуск
            return 0;
        }

    return 0;
}

void demonBody(){
    //имя устройства (веб камеры)
    string device_name="/dev/video0";

    //чтение настроек
    settingDemon *pSetting=new settingDemon;
    if(pSetting->openSetting("/home/biosoftdeveloper/etc/mydemon.conf"))
        pSetting->readSetting();
    else
        cout<<"error open setting file, set default setting"<<endl;

    string sPath=pSetting->path();
    int iTime=pSetting->time();
    pSetting->closeSetting();

    videodevice *pVideoDevice=new videodevice;
    //открывае дескриптор камеры
    pVideoDevice->openDevice(device_name);

    //настраиваем путь
    pVideoDevice->setPath(sPath);

    //сохраняемкадр с камеры
    pVideoDevice->getFrame(iTime);

    //закрываем дескриптор устройства
    pVideoDevice->closeDevice();
}
\\
\\
settingdemon.cpp

include "settingdemon.h"

#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
```

```
#include <vector>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#define SETTING_TIME    5
#define SETTING_PATH    "home/dimitr/temp/"

struct strItem{
    char *name;
    char *value;
};

vector <strItem*> vSetting;

settingDemon::settingDemon(){
    isOpen=false;
}

bool settingDemon::openSetting(string fileSetting){
    //открываем файл с настройками
    fs=fopen(fileSetting.c_str(), "r");
    if(fs==NULL){   //если не удалось открыть (нет файла, доступа)
        cout<<"Cannot open '"<<fileSetting<<"'";
        isOpen=false;
        return false;
    }

    isOpen=true;
    return true;
}

void settingDemon::closeSetting(){
    if(fs!=NULL){
        isOpen=false;
        fclose(fs);
    }
}

void settingDemon::readSetting(){
    if(!isOpen)
        return;

    //очищаем вектор от старых данных и уст. указатель на начало файла
```

```
    vSetting.clear();
    fseek(fs, 0, SEEK_SET);

    //читаем файла по строчно
    ssize_t read;
    size_t len=0;
    char *line=NULL;

    while((read=getline(&line, &len, fs))!=-1){
        //разбиваем строку на параметр и значение
        string sTemp=line;
        int iIndex=sTemp.find("=");
        if(iIndex>0){
            //сохраняем в вектор считанные данные
            int sizeValue=sTemp.size()-iIndex-2;

            strItem *pItem=new strItem;
            pItem->name=new char[iIndex+1];
            pItem->name[iIndex]=0;
            pItem->value=new char[sizeValue];
            pItem->value[sizeValue]=0;

            sTemp.copy(pItem->name, iIndex, 0);
            sTemp.copy(pItem->value, sizeValue, iIndex+1);
            vSetting.push_back(pItem);
        }
    }
}

string settingDemon::path(){
    string sTemp=SETTING_PATH;

    int index;
    if((index=findItem("PATH"))!=-1){
        strItem *pItem=vSetting[index];
        sTemp=pItem->value;
    }

    return sTemp;
}

int settingDemon::time(){
    int iTemp=SETTING_TIME;
```

```
    int index;
    if((index=findItem("TIME"))!=-1){
        strItem *pItem=vSetting[index];
        iTemp=atoi(pItem->value);
    }

    return iTemp;
}

int settingDemon::findItem(string name){
    //поиск в векторе необходимого параметра
    for(int i=0; i<vSetting.size(); i++){
        strItem *pItem=vSetting.at(i);
        if(strcmp(pItem->name, name.c_str())==0)
            return i;
    }

    return -1;
}
\\
\\
videodevice.cpp

#include <sys/ioctl.h>          //ioctrl()
#include <linux/videodev2.h>    //struct v4l2_capability
#include <unistd.h>             //close()

#include <sys/mman.h>
#include <sstream>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>


#include "videodevice.h"
#include "jpeglib.h"

#define IMAGE_WIDTH     800
#define IMAGE_HEIGHT    600

videodevice::videodevice(){
}
```

```
void videodevice::errno_exit(string err_str){
    cout<<"\""<<err_str<<"\": "<<errno<<", "<<strerror(errno)<<endl;
    exit(EXIT_FAILURE);
}

int videodevice::xioctl(int fd, int request, void *arg){
    int r;

    r=ioctl(fd, request, arg);
    if(r==-1){
        if(errno==EAGAIN)
            return EAGAIN;

        stringstream ss;
        ss<<"ioctl code "<<request<<" ";

        errno_exit(ss.str());
    }

    return r;
}

void videodevice::openDevice(string dev_name){
    fileDevicePath=dev_name;

    fd=open(fileDevicePath.c_str(), O_RDWR /* required */ | O_NONBLOCK, 0)
    if(fd==-1){
        stringstream str;
        str << "Cannot open '" << fileDevicePath << "'";
        errno_exit(str.str());
    }

    cout<<"Open device"<< fileDevicePath<<endl;
}

void videodevice::viewInfoDevice(){
    //запрос к драйверу устройства на получения информации
    struct v4l2_capability device_params;
    if(ioctl(fd, VIDIOC_QUERYCAP, &device_params)==-1){
      printf ("\"VIDIOC_QUERYCAP\" error %d, %s\n", errno, strerror(errno)
      exit(EXIT_FAILURE);
    }
```

```
    //выводи информацию о камере
    printf("driver : %s\n",device_params.driver);
    printf("card : %s\n",device_params.card);
    printf("bus_info : %s\n",device_params.bus_info);
    printf("version : %d.%d.%d\n",
         ((device_params.version >> 16) & 0xFF),
         ((device_params.version >> 8) & 0xFF),
         (device_params.version & 0xFF));
    printf("capabilities: 0x%08x\n", device_params.capabilities);
    printf("device capabilities: 0x%08x\n", device_params.device_caps);

    //format
    struct v4l2_fmtdesc fmtdesc;
    fmtdesc.index=0;
    fmtdesc.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
    while(ioctl(fd, VIDIOC_ENUM_FMT, &fmtdesc)==0){
        cout<<"image format"<<endl;
        cout<<"pixelformat: "<<fmtdesc.description<<endl;
        fmtdesc.index++;
    }
}

void videodevice::setFormatcam(){
    struct v4l2_format fmt;
    fmt.type                = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    fmt.fmt.pix.width       = IMAGE_WIDTH;
    fmt.fmt.pix.height      = IMAGE_HEIGHT;
    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUV422P;
    //fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;

    if(-1==xioctl(fd, VIDIOC_S_FMT, &fmt))
        errno_exit("VIDIOC_S_FMT");
}

void videodevice::closeDevice(){
    if(close(fd)==-1)
        errno_exit("close");
    fd=-1;

    cout<<"Close device "<<fileDevicePath<<endl;
}

void videodevice::getFrame(int iTime){
    setFormatcam();
```

```
    initMMAP();

    long int i=1;
    while(true){
        startCapturing();

        string sFile;
        std::ostringstream os;
        os<<i;
        sFile=os.str();

        //=std::to_string(i);
        for(;;){
            if(readFrame(sPath+sFile))
               break;
        }
        ++i;
        stopCapturing();

        cout<<"next frame to "<<iTime<<" sec, frame - "<<i<<endl;
        sleep(iTime);
    }

    freeMMAP();
}

void videodevice::setPath(string sPath){
    this->sPath=sPath;
}

void videodevice::initMMAP(){
    //Инициализация буфера
    struct v4l2_requestbuffers req;
    req.count=1;
    req.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
    req.memory=V4L2_MEMORY_MMAP;

    xioctl(fd, VIDIOC_REQBUFS, &req);

    //выделить память
    devbuffer=(buffer*)calloc(req.count, sizeof(*devbuffer));

    //отоброжаем буфер на память
    struct v4l2_buffer buf;
```

```
    buf.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.index=0;

    xioctl(fd, VIDIOC_QUERYBUF, &buf);

    //отоброзить память устройства в оперативную память
    devbuffer->length=buf.length;
    devbuffer->start=mmap(NULL, buf.length, PROT_READ | PROT_WRITE, MAP_SH
    if(devbuffer->start==MAP_FAILED)
        errno_exit("mmap");

    cout<<"Init mmap"<<endl;
}

void videodevice::startCapturing(){
    struct v4l2_buffer buf;
    buf.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory=V4L2_MEMORY_MMAP;
    buf.index=0;

    //буфер в очередь обработки драйвером устройства
    xioctl(fd, VIDIOC_QBUF, &buf);

    //камеру в режим захвата
    enum v4l2_buf_type type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
    xioctl(fd, VIDIOC_STREAMON, &type);

    cout<<"Start capturing"<<endl;
}

bool videodevice::readFrame(string file_name){
    struct v4l2_buffer buf;
    buf.type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory=V4L2_MEMORY_MMAP;

    //освобождаем буфер
    if(xioctl(fd, VIDIOC_DQBUF, &buf)==EAGAIN)
        return false;

    buffer *temp=devbuffer;

    //сохраняем в файл RAW
    string rawFile=file_name+string(".raw");
    FILE *out_file=fopen(rawFile.c_str(), "w");
```

```
    fwrite(temp->start, temp->length, 1, out_file);
    fclose(out_file);

    //сохраняем фото в jpg
    cout<<"save jpeg"<<endl;
    string jpegFile=file_name+string(".jpeg");
    savetojpeg(jpegFile.c_str());

    return true;
}

unsigned char* videodevice::YUVtoRGB(unsigned char *buffer){
    unsigned char *rgbBuffer=(unsigned char*)malloc(IMAGE_WIDTH*IMAGE_HEIG

    char lnPixel=2;

    for(int i=0; i<IMAGE_HEIGHT/8; i++){                                    //блок
        for(int j=0; j<IMAGE_WIDTH/8; j++){                                 //блок
            int imgData=((i*8)*IMAGE_WIDTH+j*8)*lnPixel;                    //нака
            for(int k=0; k<8; k++){                                         //пикс
                for(int l=0; l<8; l++){                                     //пикс
                    double Y=buffer[imgData+k*IMAGE_WIDTH*lnPixel+l*lnPixe
                    double Cb=buffer[imgData+k*IMAGE_WIDTH*lnPixel+l*lnPix
                    double Cr=buffer[imgData+k*IMAGE_WIDTH*lnPixel+l*lnPix

                    double r=Y+Cr-128;
                    double b=Y+Cb-128;
                    double g=Y-r-b;

                    if(r>255) r=255;
                    else if(r<0) r=0;
                    if(g>255) g=255;
                    else if(g<0) g=0;
                    if(b>255) b=255;
                    else if(b<0) b=0;

                    rgbBuffer[imgData+k*IMAGE_WIDTH*lnPixel+l*lnPixel]=r;
                    rgbBuffer[imgData+k*IMAGE_WIDTH*lnPixel+l*lnPixel+1]=g
                    rgbBuffer[imgData+k*IMAGE_WIDTH*lnPixel+l*lnPixel+2]=b
                }
            }
        }
    }
```

```
    return rgbBuffer;
}

void videodevice::savetojpeg(const char *filename){
    struct jpeg_compress_struct cinfo;
    struct jpeg_error_mgr jerr;

    FILE *outfile;                 //выходной файл jpeg
    JSAMPROW row_pointer[1];       //указатель на JSAMPLE

    //инициализация jpeg компресора
    cinfo.err=jpeg_std_error(&jerr);
    jpeg_create_compress(&cinfo);

    //открытия файла для сохраннения
    if((outfile=fopen(filename, "wb")) == NULL) {
        fprintf(stderr, "can't open %s\n", filename);
        exit(1);
    }

    jpeg_stdio_dest(&cinfo, outfile);
    cinfo.image_width=IMAGE_WIDTH;         //размер изображения
    cinfo.image_height=IMAGE_HEIGHT;
    cinfo.input_components=3;              //количество цветовых компанент
    cinfo.in_color_space=JCS_YCbCr;          //цветовая схема

    //установка параметров изображения
    jpeg_set_defaults(&cinfo);

    //установка качества изображения
    jpeg_set_quality(&cinfo, 200, TRUE);

    //Начало компрессии изображения
    jpeg_start_compress(&cinfo, TRUE);

    int row_stride=IMAGE_WIDTH;

    //unsigned char *bufferRGB=YUVtoRGB((unsigned char*)devbuffer->start);
    //JSAMPLE *image_buffer=(JSAMPLE*)bufferRGB;
    JSAMPLE *image_buffer=(JSAMPLE*)devbuffer->start;
    while (cinfo.next_scanline<cinfo.image_height) {
        //запись построчна
        row_pointer[0]=&image_buffer[cinfo.next_scanline*row_stride];
        (void) jpeg_write_scanlines(&cinfo, row_pointer, 1);
```

```
    }

    jpeg_finish_compress(&cinfo);
    fclose(outfile);
    jpeg_destroy_compress(&cinfo);
}

void videodevice::stopCapturing(){
    enum v4l2_buf_type type;

    type=V4L2_BUF_TYPE_VIDEO_CAPTURE;
    xioctl(fd, VIDIOC_STREAMOFF, &type);

    cout<<"stop Capturing"<<endl;
}

void videodevice::freeMMAP(){
    if(munmap(devbuffer->start, devbuffer->length)==-1)
        errno_exit("munmap");

    free(devbuffer);

    cout<<"free mmap"<<endl;
}
```