ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

старший преподаватель		М. Д. Поляк
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О КУРСОВОЙ РАБОТЕ №8

СИСТЕМА СЛЕЖЕНИЯ.

по курсу: ОПЕРАЦИОННЫЕ СИСТЕМЫ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4236		А. К. Панин
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

Цель работы:

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

Индивидуальное задание:

Реализовать драйвер веб-камеры (демон), который делает снимок с камеры каждые 30 секунд и сохраняет его на диске в одном из популярных сжатых графических форматов (JPEG, TIFF, и т.п.). Предусмотреть возможность настройки временного интервала между снимками и пути, по которому сохраняются файлы.

Сравнение с аналогами:

В данной работе реализован драйвер веб-камеры для операционной системы Linux, который периодически делает снимки с камеры и сохраняет их в виде файлов в популярных графических форматах, таких как JPEG. Для выполнения аналогичной задачи существует несколько готовых решений, таких как программы fswebcam, mjpg-streamer, а также различные сторонние библиотеки для работы с видеоустройствами.

fswebcam

fswebcam является популярным инструментом для захвата изображений с веб-камеры в Linux. Он поддерживает множество параметров настройки, таких как выбор разрешения, формата изображения и частоты съемки. В отличие от реализованного в данной работе драйвера, fswebcam не является демоном и не предоставляет функционала для автоматического выполнения задачи в фоновом режиме. Однако, fswebcam позволяет настраивать частоту съемки с помощью параметров командной строки и сохранять изображения в различных форматах, включая JPEG.

mjpg-streamer

mjpg-streamer — это утилита для потоковой передачи видеопотока с веб-камеры. Она позволяет организовать поток в формате MJPEG, который может быть доступен через веб-интерфейс. Несмотря на свою популярность в области видеонаблюдения, mjpg-streamer не предназначен для периодического создания снимков и их сохранения. В отличие от выбранного решения, которое работает в фоновом режиме как демон и сохраняет изображения по расписанию, mjpg-streamer требует постоянного подключения к сети и настроек на стороне сервера.

Использование библиотеки Video4Linux2

Программные решения, использующие Video4Linux2 (V4L2), такие как v4l-utils, позволяют взаимодействовать с видеоустройствами на уровне ядра, предоставляя мощные возможности для захвата и обработки видеопотока. Однако, они не всегда предлагают возможности для реализации сложных сценариев, таких как автоматическое создание снимков с заданным интервалом. В отличие от них, разработанное решение сосредоточено на автоматизации процесса съемки изображений с заданным интервалом времени и их сохранении в нужном формате.

Основные отличия

Основное отличие предлагаемого решения от существующих аналогов заключается в его ориентации на автоматическое выполнение задачи с заданным интервалом времени и сохранение снимков на диск. Разработанный драйвер работает в фоновом режиме, используя стандартный механизм демона и конфигурационный файл для настройки параметров работы, таких как путь сохранения и интервал между снимками.

Также, в отличие от большинства аналогичных решений, разработанный драйвер позволяет сохранять изображения в сжатом формате JPEG, что экономит место на диске и упрощает использование полученных снимков.

Преимущества и недостатки

Преимущества реализованного решения:

- Функционал демона для автоматической работы в фоновом режиме.
- Возможность настройки интервала между снимками и пути сохранения.
- Простота в использовании и конфигурировании через конфигурационный файл.
- Поддержка формата JPEG, что удобно для дальнейшего использования изображений.

Недостатки:

- Ограниченная поддержка только одного формата изображения (JPEG).
- Нет поддержки потоковой передачи видео, как в mjpg-streamer.
- Взаимодействие только с одним видеоустройством (/dev/video0).

Таким образом, предлагаемое решение представляет собой удобный инструмент для периодической съемки и сохранения изображений с веб-камеры, с преимуществами в плане автоматизации и конфигурируемости, однако оно ограничено в плане поддержки форматов и функционала по сравнению с более универсальными инструментами, такими как fswebcam или mjpg-streamer.

Техническая документация:

В данном разделе описана пошаговая инструкция по установке и использованию разработанного программного обеспечения.

Предварительные требования

Перед установкой необходимо убедиться, что на системе выполнены следующие условия:

- Операционная система Linux, использующая систему инициализации SystemD (Ubuntu, Debian, CentOS и другие). В инструкции приведены команды для дистрибитивов, основанных на Debian.
- Установлены необходимые пакеты:
  - build-essential для компиляции исходного кода.
  - libjpeg-dev для работы с изображениями в формате JPEG.
  - v4l-utils для работы с видеоустройствами.
- Устройство веб-камеры подключено к компьютеру и доступно через /dev/video0.

Установка

Для установки программного обеспечения выполните следующие шаги.

1) Клонируйте репозиторий с исходным кодом:

   ```
   git clone https://github.com/Bolotnik-ss/os-option8.git
   cd os-option8/src
   ```

2) Настройте директорию для сохранения снимков и интервал съемки с помощью файла option8.conf. В этом файле можно настроить следующие параметры:

   - save_path — путь к каталогу, где будут сохраняться снимки. Пример:

     ```
     save_path = /tmp/snapshots
     ```

   - interval — временной интервал между снимками в секундах. Пример:

     ```
     interval = 30
     ```

   Если конфигурация не задана, будут использоваться значения по умолчанию:

   - save_path = /tmp/snapshots
   - interval = 30

3) Соберите программу с помощью make:

   make

4) Установите программу в систему:

   sudo make install

   Это установит исполняемый файл в /usr/local/bin, сервисный файл в /etc/systemd/system и конфигурационный файл в /etc. При этом сервис запустится самостоятельно.

Конфигурация

После установки драйвера, его конфигурация будет храниться в файле /etc/option8.conf. Работа с файлом аналогична процессу конфигурации перед установкой драйвера.

Использование

После установки и настройки программы, сервис будет автоматически делать снимки с камеры с интервалом, заданным в конфигурационном файле, и сохранять их в указанной папке. Также сервис будет настроен на автоматический запуст после запуска системы.

Для остановки или перезапуска сервиса используйте следующие команды:

- Остановить сервис:

  sudo systemctl stop option8.service

- Запустить сервис:

  sudo systemctl start option8.service

- Перезапустить сервис:

  sudo systemctl restart option8.service

- Убрать сервис из автозапуска:

  sudo systemctl disable option8.service

- Добавить сервис в автозапуск:

  sudo systemctl enable option8.service

- Посмотреть статус сервиса:

  sudo systemctl status option8.service

Снимки будут сохраняться в каталоге, указанном в конфигурационном файле, с именами в формате:

snapshot_YYYYMMDD_HHMMSS.jpg

Удаление

Для удаления программы удалите сервис и файлы программы:

sudo make uninstall

Ошибки и их устранение

- Если программа не может открыть видеоустройство /dev/video0, убедитесь, что камера подключена и правильно настроена в системе.
- Если программа не запускается или не сохраняет изображения, проверьте журнал ошибок с помощью команды:

  journalctl -u option8.service

Выводы:

В ходе выполнения курсового проекта были достигнуты следующие результаты:

1) Изучены принципы взаимодействия с ядром операционной системы Linux и работы драйверов устройств.
2) Реализован пользовательский драйвер, осуществляющий захват изображений с вебкамеры через интерфейс Video4Linux2 (V4L2) и их сохранение в формате JPEG.
3) Обеспечена возможность настройки параметров работы программы, таких как временной интервал между снимками и путь сохранения файлов, через конфигурационный файл.
4) Создана документация, описывающая процесс установки, настройки и использования разработанного программного обеспечения.

5) Получены навыки работы с системой сборки make, системными службами SystemD и управления процессами в ОС Linux.

Проект достиг поставленных целей, включая реализацию функционала захвата и сохранения изображений с возможностью конфигурирования. Разработанное решение успешно демонстрирует возможности взаимодействия с аппаратным обеспечением через V4L2 и применяет базовые методы обработки изображений.

Данный курсовой проект позволяет получить базовые знания и навыки, необходимые для разработки драйверов и приложений, работающих с периферийными устройствами в ОС Linux.

Приложение. Листинги файлов:

option8.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <signal.h>
#include <linux/videodev2.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <errno.h>
#include <jpeglib.h>

#define DEFAULT_PATH "/tmp/snapshots"
#define DEFAULT_INTERVAL 30
#define CONFIG_FILE "/etc/option8.conf"

static int running = 1;
static int interval = DEFAULT_INTERVAL;
static char save_path[256] = DEFAULT_PATH;

void handle_signal(int sig) {
    running = 0;
}

void daemonize() {
    pid_t pid = fork();
    if (pid < 0) {
        perror("Не удалось выполнить fork");
        exit(EXIT_FAILURE);
    }
```

```
    if (pid > 0) {
        exit(EXIT_SUCCESS);
    }

    if (setsid() < 0) {
        perror("Не удалось создать новую сессию (setsid)");
        exit(EXIT_FAILURE);
    }

    signal(SIGCHLD, SIG_IGN);
    signal(SIGHUP, SIG_IGN);

    pid = fork();
    if (pid < 0) {
        perror("Не удалось выполнить fork");
        exit(EXIT_FAILURE);
    }
    if (pid > 0) {
        exit(EXIT_SUCCESS);
    }

    umask(0);
    chdir("/");

    for (int x = sysconf(_SC_OPEN_MAX); x >= 0; x--) {
        close(x);
    }
}

void load_config() {
    FILE *config_file = fopen(CONFIG_FILE, "r");
    if (!config_file) {
        perror("Не удалось открыть конфигурационный файл");
        return;
    }

    char line[256];
    while (fgets(line, sizeof(line), config_file)) {
        line[strcspn(line, "\n")] = '\0';

        if (line[0] == '\0' || line[0] == '#') {
            continue;
        }

        char key[128], value[128];
        if (sscanf(line, "%127s = %127s", key, value) == 2) {
            if (strcmp(key, "save_path") == 0) {
                strncpy(save_path, value, sizeof(save_path) - 1);
                save_path[sizeof(save_path) - 1] = '\0';
            } else if (strcmp(key, "interval") == 0) {
```

```
            interval = atoi(value);
        }
    }
}

fclose(config_file);
}

void save_image_as_jpeg(const void *buffer, size_t width, size_t height) {
    char filename[512];
    time_t now = time(NULL);
    struct tm *t = localtime(&now);

    snprintf(filename, sizeof(filename), "%s/snapshot_%04d%02d%02d_%02d%02d%02d.jpg",
             save_path, t->tm_year + 1900, t->tm_mon + 1, t->tm_mday, t->tm_hour, t->tm_min, t->tm_sec);

    FILE *file = fopen(filename, "wb");
    if (!file) {
        perror("Не удалось сохранить изображение");
        return;
    }

    struct jpeg_compress_struct cinfo;
    struct jpeg_error_mgr jerr;
    JSAMPROW row_pointer[1];

    cinfo.err = jpeg_std_error(&jerr);
    jpeg_create_compress(&cinfo);
    jpeg_stdio_dest(&cinfo, file);

    cinfo.image_width = width;
    cinfo.image_height = height;
    cinfo.input_components = 3;
    cinfo.in_color_space = JCS_RGB;

    jpeg_set_defaults(&cinfo);
    jpeg_start_compress(&cinfo, TRUE);

    unsigned char *rgb_buffer = (unsigned char *)malloc(width * height * 3);
    if (!rgb_buffer) {
        perror("Не удалось выделить память для буфера RGB");
        fclose(file);
        return;
    }

    for (size_t i = 0; i < height; i++) {
        for (size_t j = 0; j < width; j++) {
            size_t yuyv_index = (i * width + j) * 2;
            unsigned char Y = ((unsigned char *)buffer)[yuyv_index];
```

```
            unsigned char U = ((unsigned char *)buffer)[yuyv_index + 1];
            unsigned char V = ((unsigned char *)buffer)[yuyv_index + 2];

            int C = Y - 16;
            int D = U - 128;
            int E = V - 128;

            int R = (298 * C + 409 * E + 128) >> 8;
            int G = (298 * C - 100 * D - 208 * E + 128) >> 8;
            int B = (298 * C + 516 * D + 128) >> 8;

            rgb_buffer[(i * width + j) * 3] = (R < 0) ? 0 : (R > 255) ? 255 : R;
            rgb_buffer[(i * width + j) * 3 + 1] = (G < 0) ? 0 : (G > 255) ? 255 : G;
            rgb_buffer[(i * width + j) * 3 + 2] = (B < 0) ? 0 : (B > 255) ? 255 : B;
        }

        row_pointer[0] = &rgb_buffer[i * width * 3];
        jpeg_write_scanlines(&cinfo, row_pointer, 1);
    }

    jpeg_finish_compress(&cinfo);
    jpeg_destroy_compress(&cinfo);
    fclose(file);
    free(rgb_buffer);

    printf("Сохранен снимок: %s\n", filename);
}

int main(int argc, char *argv[]) {
    int fd;
    struct v4l2_capability cap;
    struct v4l2_format fmt;
    struct v4l2_requestbuffers req;
    struct v4l2_buffer buf;
    void *buffer;
    unsigned int buffer_length;

    load_config();

    daemonize();

    signal(SIGINT, handle_signal);
    signal(SIGTERM, handle_signal);

    fd = open("/dev/video0", O_RDWR);
    if (fd == -1) {
        perror("Не удалось открыть видеоустройство");
        exit(EXIT_FAILURE);
    }
```

```
if (ioctl(fd, VIDIOC_QUERYCAP, &cap) == -1) {
    perror("Не удалось запросить возможности устройства");
    close(fd);
    exit(EXIT_FAILURE);
}

memset(&fmt, 0, sizeof(fmt));
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.width = 640;
fmt.fmt.pix.height = 480;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;

if (ioctl(fd, VIDIOC_S_FMT, &fmt) == -1) {
    perror("Не удалось установить формат видео");
    close(fd);
    exit(EXIT_FAILURE);
}

memset(&req, 0, sizeof(req));
req.count = 1;
req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
req.memory = V4L2_MEMORY_MMAP;

if (ioctl(fd, VIDIOC_REQBUFS, &req) == -1) {
    perror("Не удалось запросить буферы");
    close(fd);
    exit(EXIT_FAILURE);
}

memset(&buf, 0, sizeof(buf));
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory = V4L2_MEMORY_MMAP;
buf.index = 0;

if (ioctl(fd, VIDIOC_QUERYBUF, &buf) == -1) {
    perror("Не удалось запросить буфер");
    close(fd);
    exit(EXIT_FAILURE);
}

buffer = mmap(NULL, buf.length, PROT_READ | PROT_WRITE, MAP_SHARED, fd, buf.m.of
buffer_length = buf.length;

if (buffer == MAP_FAILED) {
    perror("Не удалось отобразить буфер в память");
    close(fd);
    exit(EXIT_FAILURE);
}
```

```
    if (mkdir(save_path, 0755) == -1 && errno != EEXIST) {
        perror("Не удалось создать каталог для сохранения");
        munmap(buffer, buffer_length);
        close(fd);
        exit(EXIT_FAILURE);
    }

    while (running) {
        if (ioctl(fd, VIDIOC_QBUF, &buf) == -1) {
            perror("Не удалось поставить буфер в очередь");
            break;
        }

        if (ioctl(fd, VIDIOC_STREAMON, &buf.type) == -1) {
            perror("Не удалось запустить захват");
            break;
        }

        if (ioctl(fd, VIDIOC_DQBUF, &buf) == -1) {
            perror("Не удалось извлечь буфер из очереди");
            break;
        }

        save_image_as_jpeg(buffer, fmt.fmt.pix.width, fmt.fmt.pix.height);

        sleep(interval);
    }

    munmap(buffer, buffer_length);
    close(fd);

    return 0;
}
```

option8.service:

```
[Unit]
Description=option8

[Service]
ExecStart=/usr/local/bin/option8
Type=forking
KillMode=process
SyslogIdentifier=smart-test
SyslogFacility=daemon
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Makefile:

```
CC=gcc
CFLAGS=-Wall -g
LDFLAGS=-ljpeg
SRC=option8.c
BINARY=option8
SERVICE=option8.service
CONFIG=option8.conf
BINARY_DIR=/usr/local/bin
SERVICE_DIR=/etc/systemd/system
CONFIG_DIR=/etc

.PHONY: all clean install uninstall

all: $(BINARY)

$(BINARY): $(SRC)
$(CC) $(CFLAGS) $(SRC) -o $(BINARY) $(LDFLAGS)

install: all
install -m 0755 $(BINARY) $(BINARY_DIR)
install -m 0644 $(SERVICE) $(SERVICE_DIR)
install -m 0644 $(CONFIG) $(CONFIG_DIR)
systemctl daemon-reload
systemctl enable option8.service
systemctl start option8.service

uninstall:
systemctl stop option8.service
systemctl disable option8.service
rm -f $(CONFIG_DIR)/$(CONFIG)
rm -f $(SERVICE_DIR)/$(SERVICE)
rm -f $(BINARY_DIR)/$(BINARY)
systemctl daemon-reload

clean:
rm -f $(BINARY)
```

option8.conf:

```
save_path = /tmp/snapshots
interval = 30
```