УДК 004.9

МНОГОКАНАЛЬНАЯ СИСТЕМА ПОЛУЧЕНИЯ ВИДЕОИНФОРМАЦИИ НА ПРОГРАММНОЙ ПЛАТФОРМЕ LINUX

Иванов Ю.А., Вовк С.М.
Днепропетровский национальный университет им. О. Гончара, Украина

Представлен подход к разработке многоканальной системы съема видеоданных на основе программной платформы Linux. Представлено программное обеспечение, которое обеспечивает подключение нескольких видеокамер и может быть использовано во встраиваемых устройствах для решения задач одновременной обработки видеоданных по нескольким каналам, а также для разработки многоканальных систем компьютерного зрения.

Общая постановка проблемы

Современные информационные системы развиваются по пути увеличения объема получаемой информации и усложнения алгоритмов ее обработки. В настоящее время все шире применяются многоканальные системы, которые позволяют существенно увеличить объем информации об объектах или процессах. В области видеоинформационных технологий стали широко применяться многоканальные (многопозиционные) системы [1]. В частности такими системами являются системы управления дорожным движением, которые фиксируют перекресток с разных ракурсов (интеллектуальный перекресток) [2], медицинские системы и др. Одной из актуальных задач можно считать разработку встроенных систем, предназначенных для применения в автомобильном транспорте, когда на переднюю панель монитора водителя выводится видеоинформация окружающей обстановки переднего, заднего и боковых планов, что позволяет осуществлять водителю сложные маневры при выезде со стоянки или при парковке [3]. Данная работа посвящена построению многоканальной видеоинформационной системы, которая может быть применена во встраиваемых системах на базе ядра Linux. При этом для обеспечения эффективной работы системы в реальном масштабе времени используется сборка системы из таких наиболее важных программных компонентов, как ядро Linux с модулем uvcvideo, Xorg server, библиотеки libv4l2 и Qt Framework.

Постановка задачи

Традиционный подход к разработке многоканальных систем съема данных основан на применении многопоточного программирования, где каждый поток отвечает за свой канал системы, а главный поток осуществляет координацию этих потоков и функционирование всей системы в целом. При увеличении числа потоков естественно возникает нагрузка на ядро системы и, с точки зрения системы реального времени, могут возникать ситуации недиспетчеризуемости системы из-за пропуска критических сроков обслуживания. Постановка рассматриваемой задачи заключалась в разработке и реализации многоканальной системы получения видеоинформации на базе стандартных срезов платформы Linux и исследовании временных характеристик работы многоканальной системы на базе стандартного процессора Intel Celeron 2.0GHz в зависимости от количества подключенных каналов и оптимизации программного кода.

Решение задачи и результаты исследований

Архитектура приложения, которое реализует многоканальный съем видеоданных включает главный поток и по одному потоку на каждую камеру (рис.1). В главном потоке осуществляется первичная инициализация камер и подготовка их к захвату изображения, а также вывод готовых изображений на графическую форму. Дочерний поток камеры ожидает готовности кадра, конвертирует его в формат, требуемый для вывода на форму, копирует его во временный буфер и сигнализирует

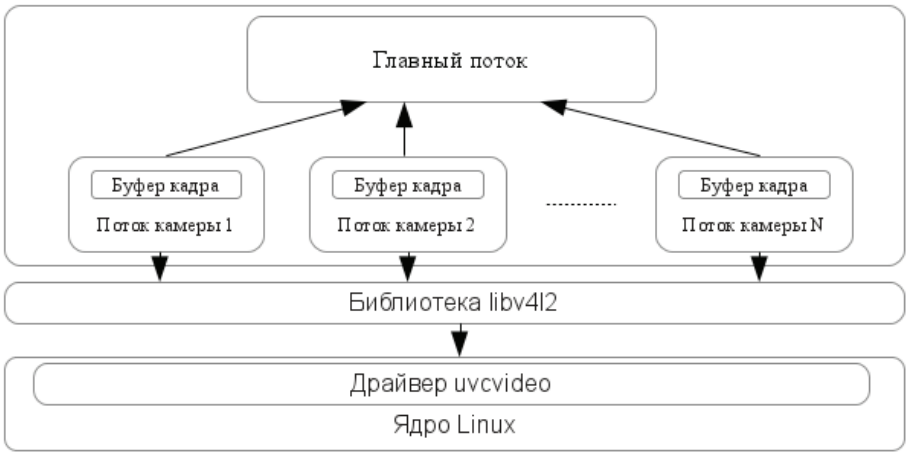

Рисунок 1. Архитектура приложения

главному потоку о готовности изображения. Для сигнализации о готовности изображения была использована технология сигналов и слотов библиотеки Qt Framework [4]. Для инициализации камер и захвата изображений использовалась библиотека libv4l2 [5, 6]. Управление камерой осуществляется драйвером uvcvideo, который входит в состав ядра Linux. Камеры были подключены через USB интерфейс [7] с помощью контроллера Nec.

Программный код потока камеры имеет следующие особенности. Класс камеры V4LCamera является наследником класса QThread. QThread представляет собой отдельный поток управляемый в рамках программы; он разделяет данных со всеми другими потоками в процессе, но выполняется самостоятельно. Поток начинает свое выполнение с функции run(). По умолчанию run() начинает цикл обработки событий вызовом метода exec(). Для создания собственного потока нужно наследовать класс Qthread и переопределить функцию run(). Далее приведена реализация функции run():

```
void V4LCamera::readFrame() {
  /* объявляем переменную buf для работы с буфером
  кадра камеры, задаем тип операции и тип памяти,
  и выполняем запрос к камере */
    struct v4l2 _ buffer buf; CLEAR (buf);
    buf.type = V4L2 _ BUF _ TYPE _ VIDEO _ CAPTURE;
    buf.memory = V4L2 _ MEMORY _ MMAP;
    if (-1 == xioctl (&fd, VIDIOC _ DQBUF, &buf)) return;
    assert (buf.index < nBuffers);
    /* конвертируем изображение в нужный нам формат */
    if (v4lconvert _ convert(v4lconvertData, &srcFmt, &fmt,
      (unsigned char*)buffers[buf.index].start,
       buf.bytesused, image, fmt.fmt.pix.sizeimage) < 0) {
       if (errno != EAGAIN)
         qDebug(«Error: Couldn’t convert image»); }
    if (-1 == xioctl (&fd, VIDIOC _ QBUF, &buf)) return; }

void V4LCamera::run() {
    while (running) {
    /* объявляем структуру для работы с сокетом,
    выставляем таймаут ожидания кадра 2 секунд */
        fd _ set fds; struct timeval tv;
        FD _ ZERO (&fds); FD _ SET (fd, &fds);
        tv.tv _ sec = 2; tv.tv _ usec = 0;
    /* если за 2 секунды кадр не получен,
    то завершаем работу потока */
        if (0 >= select (fd + 1, &fds, NULL, NULL, &tv))
          return;
        //читаем и конвертируем изображение
```

```
        readFrame();
        //инициируем сигнал
        emit imageReady(image);
    } }
```

Для передачи кадра из потока камеры в главный поток используется механизм сигналов и слотов. Для этого используется метод void imageReady(uint8_t *) - функция сигнал; она не нуждается в реализации и в качестве аргумента принимает указатель на изображение. В качестве слота выступает метод класса CameraWidget - void paintImage(uint8_t *). Метод имеет аргумент такого же типа, как и сигнал. Реализация слота и метода для рисования изображения выполняются следующим образом:

```
void CameraWidget::paintEvent(QPaintEvent *) {
  /* это метод вызывается при изменении размеров виджета,
  а так же при вызове метода update() */
    QPainter painter(this);
    if (image == NULL) {
    /* если буфер кадра пуст  - рисуем надпись
       "Camera output" синего цвета */
      painter.setPen(Qt::blue);
      painter.setFont(QFont(«Arial», 30));
      painter.drawText(rect(), Qt::AlignCenter,
                       «Camera output»);
    } else {
    /* иначе создаем из чистых данных кадра объект
       класса QImage и рисуем его на фиджете */
      QImage img(image, 320, 240, QImage::Format _ RGB888);
      painter.drawImage(QPoint(0,0), img);
} }

void CameraWidget::paintImage(uint8 _ t * image) {
  /* копируем изображение во временный буфер виджета
     и вызываем метод обновления виджета */
    this->image = image;
    update(); }
```

Сигнал и слот связываются методом connect, который можно вызвать так:

```
connect(camera, SIGNAL(imageReady(uint8 _ t*)), this, SLOT(paintImage(uint8 _ t*)));
```

При проведении исследования было выявлено, что при попытке получения видеоинформации с разрешением 640x480 точек 30 кадров в секунду от двух и более камер одновременно захват

Рисунок 2. Внешний вид приложения

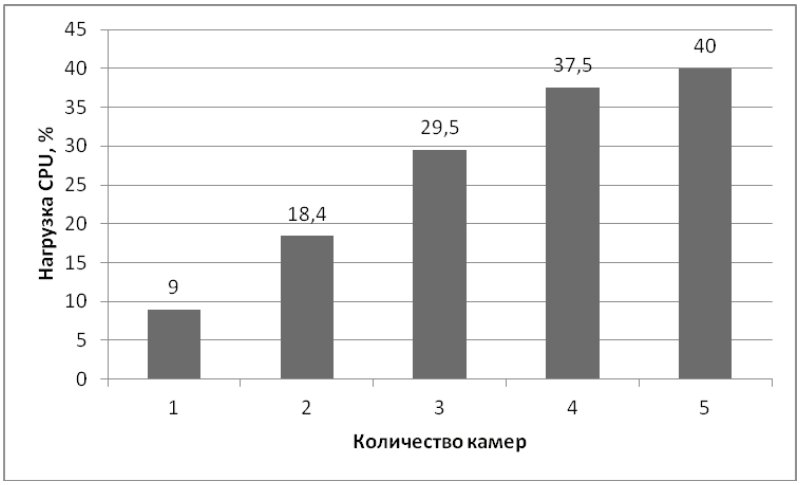

Рисунок 5. Результаты тестирования

4

происходил только с одной камеры, остальные же при инициализации выдавали ошибку и отказывались работать. Анализ этой ситуации показал, что драйвер веб-камеры uvcvideo резервирует всю полосу пропускания USB 2.0, хотя для этого нет особой необходимости. Скорость видеопотока для разрешения 640x480 точек 30 кадров в секунду в формате YUYV составляет ≈140Mbit/s, а теоретическая пропускная способность шины USB 2.0 — 480Mbit/s. Поэтому эта ситуация может быть разрешена двумя способами: либо добавлением USB контроллеров, либо уменьшением разрешения изображения. В дальнейшем был использован второй способ, когда разрешение изображения было уменьшено до 320x240 точек 30 кадров в секунду.

Для увеличения производительности приложения не использовались специализированные библиотеки по работе с камерами и обработке изображений такие как OpenCV [8]. Конвертирование изображения производилось средствами библиотеки libv4lconvert, которая входит в состав библиотеки libv4l2. Это позволило увеличить производительность примерно в 3 раза. Так же оптимизация программного кода проводилась средствами компилятора.

Для проведения тестов использовался компьютер с процессором Intel Celeron 2.0GHz, веб-камеры Logitech QuickCam Pro 5000, операционная система OpenSUSE Linux (ядро версии 2.6.37), Qt Framework версии 4.7.1, libv4l2 версии 0.8.5, компилятор gcc 4.5.1. Результаты тестирования приведены на рис. 3.

Выводы

Разработанное программное обеспечение обеспечивает подключение нескольких видеокамер и может быть использовано во встраиваемых устройствах для решения задач одновременной обработки видеоданных по нескольким каналам, а также для разработки многоканальных систем компьютерного зрения. Тестирование системы показало, что при подключении 5 камер через USB интерфейс загрузка процессора достигает 40%.

Литература

[1] Форсайт Д. Компьютерное зрение. Современный подход. / Д. Форсайт, Ж. Понс М.: Издательский дом «Вильямс», 2004. — 928 с.

[2] Интеллектуальный Перекресток. Интеллектуальный перекресток от «Интегра-С». Электронный ресурс. Режим доступа: http://www.secuteck.ru/articles2/videonabl/intellektualnii-perekrestok-integra-s

[3] BMW X3: Видеосистемы. Материал из официального сайта BMW. Электронный ресурс. Режим доступа: http://www.bmw.ru/ru/ru/newvehicles/x/x3/2010/showroom/comfort/rear_view_camera.html

[4] Библиотека Qt Framework. Материал из официального сайта Qt. Электронный ресурс. Режим доступа: http://qt.nokia.com/

[5] Библиотеки libv4l и v4l-utils. Материал из домашней страницы проекта. Электронный ресурс. Режим доступа: http://freshmeat.net/projects/libv4l

[6] Библиотека пространства пользователя libv4l. Материал с сайта «Television with linux» Электронный ресурс. Режим доступа: http://linuxtv.org/downloads/v4l-dvb-apis/libv4l.html

[7] Агуров П. В. Интерфейсы USB. Практика использования и программирование. / П. В. Агуров. - СПб: БХВ-Петербург, 2005. – 576 с.

[8] Библиотека OpenCV. Материал с официального сайта. Электронный ресурс. Режим доступа: http://opencv.willowgarage.com/wiki/