Video4Linux programming introduction

Introduction

Video4Linux is the name of the linux‘s video capture software interface. Webcameras, capture cards, tv cards, and this kind of devices were grouped into the video capture category, and a API (application programming interface) was developed to deal with it.

Due to these device‘s nature, the usual Unix device abstraction had to be a little bit extended. Usually, you can access every device, just doing an open() in a /dev/file, and reading/writing some bytes.

Video devices work this way, except that you will find a lot more of ioctl()s to deal with some options, and a more developed mmap() interface to deal with frame copy, instead of read()ing each byte. This is very important to those who need to do a very fast capture sequence.

This workflow initially is a kinda hard to understand, but after that, you will find yourself worrying about how to implement a faster capture scheme than which ioctl to use. It‘s really interesting. BTW, there is a video4windows API, but its not compatible, sometimes refered as v4w or vfw.

Motivation

I‘ve been developing videodog, since may/2000 and one year later, decided to open it to the community. My main goal was a simple capture utility, with none or a little dependencies. All other applications were based in imlib, gtk, or other libraries, what makes cool screenshots, but for a real working situation can be a pain, due to the need of all dependencies being installed.

So, I started using only jpeglib, but its not mandatory, you can use other formats and do a external jpeg or png conversion with any other tool. Thats how unix works, small components working togheter. VideoDog can be work in small computers, like a video surveillance system, a testbed for image processing algorithms, or the videocapture interface for your script/program.

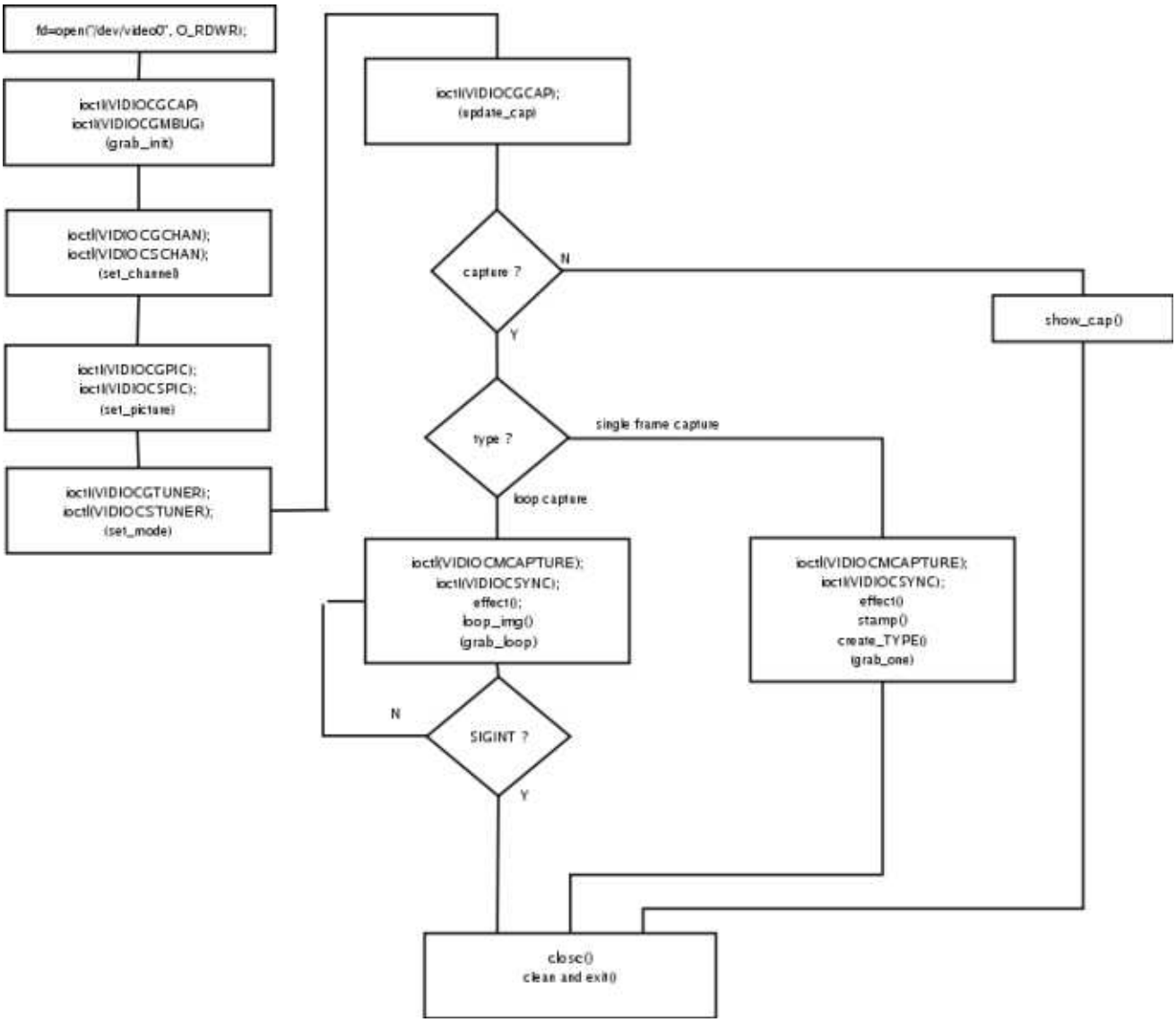
Since the begginin it has wvolved due to my demands and suggestions from users all around the world. After jpeg encoding, I‘ve implemented time stamp, using a small and simple charset. There was no need to different fonts, so, no need to depend on external libraries, such libttf.

Design decisions like that are meant to keep it simple yet useful. The main feedback I receive about VideoDog is that it is almost plug and play, and production ready. Umtil now, http://webcam.mit.edu has been using it to power their webcam site. This is only one application, I‘ve got reports about scientific work being done using VideoDog as testbed and plataform, since medical applications to computer vision testes. Most users dont want to deal with complicated code, which many times are result more of developer‘s ego to show C or C++ skills than design decisions.

Anyway, I will be using VideoDog in its current version to show living examples of code, a ancient release, to show a simple capture program and a small but complete full speed capture code.

Simple capture block diagram

As everything in programming, can be done in many ways. This is how VideoDog works



Program structure

VideoDog has a modular approach to deal with the V4L API. After parse its command line or config files options, setup the device and capture, either a single frame, or a looping mode capture, which it tries to achieve the highest capture speed possible, which can vary due to system configuration. Basic initialization scheme is as follows:

```
open
VIDIOCGCAP
VIDIOCGCHAN
norm
channel
VIDIOCSCHAN
VIDIOCGPICT
set grab_buf
mmap
VIDIOCSPICT
Proceed capture...
```

As said before, many operations are triggered by IOCTLs, from image size and device capabilities, to the capture/cyn mechanics. By the way, this is the point where many developers get confused (including me).

From our point of view, frame capture happnes in two steps:

1) Capture from video input to video capture device

2) Transfer from device to user space accessible memory

So, to do a loop capture, you need to sync the MAXBUF possible buffers (default are two for video boards, but can be different. Get them with VIDIOCGMBUF). First, use VIDIOCMCAPTURE to put all them in the capture queue. After that, sync the VIDIOCSYNC with VIDIOCMCAPTURE in a loop to, while the drivers capture to buffer one, you can read buffer two, and so on, while capturing to buffer two, buffer one be avaliable to you.

This is the heart of the multiple frames capture. Sync'ing these two IOCTLs, and working properly with them, helps to achieve maximum framerates, like almost 30 frames per second.

Instead of read()ing bytes from the device, we use the mmap() syscall. Mmap, or memory map, asks to map a file or device, or any object, to memory, since a given address. Why is that ? Because this makes possible to memcpy() the intare image once, instead of reading serially from the device. Many v4l drivers have evolved their mmap interfaces more than the read() one.

The mmap mechanics inside v4l context is very interesting. Althought we have mapped mmap once, the driver uses an offset value to the start address, to put the new sync'ed image. So, extending the ioctl explanation above, we would **VIDIOCMCAPTURE** to buffer, and buffer+image_lenght (width * height * depth in a RGB image), as putting them in a capture queue. After that, inside de loop, while we **VIDIOCSYNC**, buffer, we would be using **VIDIOCMCAPTURE** in the second buffer, offset by image lenght, and vice-versa, while capturing to the second buffer, sync'ing the first one. It can be extended to as many buffers the driver supports. We assume that buffer mas mmap'ed previously.

This is a simple process, as seen on code, but a little hard to understand if you are not an unix programmer.

The V4L API doc has an extensive ioctl list and the struct members, so, I think its not mandatory to replicate them here.

The following code shows this capture mode.

SIMPLE CAPTURE APPLICATION USING DOUBLE BUFFERING

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <asm/types.h>
#include <linux/videodev.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <asm/types.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>
#include <signal.h>
#include <errno.h>

#define DOT fprintf(stderr,".");

struct video_picture      grab_pic;
struct video_capability   grab_cap;
/* choose channel and (NTSC, PAL-M, SECAM) */
struct video_channel      grab_vid;
struct video_mmap         grab_buf;
struct video_mbuf      grab_vm;
unsigned int              grab_fd, grab_size, frame, i, done=1;
unsigned char         *grab_data, *buf;

/* simple function to save a PNM file */

int save_pnm (char *buf, int x, int y, int depth) {
     static int inc=0;
     FILE *fp;

     char bewf[128];
     sprintf (bewf, "image-%d.pnm", inc);

     if ((fp=fopen(bewf, "w+")) == NULL) {
          perror("open");
          exit(1);
          }

     if (depth==3) fprintf(fp, "P6\n%d %d\n255\n", x, y);
     else if (depth==1) fprintf(fp, "P5\n%d %d\n255\n", x, y);

     fwrite ((unsigned char*) buf, x * y * depth, 1, fp);
     inc ++; // next name

     fclose (fp);
     }


void main (int argc, char **argv) {
     int x=320, y=240, depth=3;
     time_t inito, endo;

     grab_size= x * y * depth;

     if ((grab_fd = open("/dev/video",O_RDWR)) == -1 ) {
          perror("open videodev");
          exit(-1);
     }

     if (ioctl(grab_fd,VIDIOCGCAP,&grab_cap) == -1) {
```

```
        fprintf(stderr,"wrong device\n");
        exit(-1);
    }


    /* Query the actual buffers available */
    if(ioctl(grab_fd, VIDIOCGMBUF, &grab_vm) < 0) {
        perror("VIDIOCGMBUF");
        exit(-1);
    }

    grab_buf.height = x; /* Your own height */
    grab_buf.width  = y; /* Your own width */
    grab_buf.format = VIDEO_PALETTE_RGB24; /* Your own format */


    /* MMap all available buffers */
    grab_data = (unsigned char *)mmap(0, grab_vm.size,
PROT_READ|PROT_WRITE, MAP_SHARED, grab_fd, 0);

    if (grab_data==(unsigned char *)-1) {
        perror("mmap");
        exit(-1);
    }

    /* Queue all available buffers for capture */


    /* query about number of frames and queue them*/

    fprintf (stderr,"buffers: %d - size: %d", grab_vm.frames,
grab_vm.size);

    for(frame=0; frame<grab_vm.frames; frame++) {
        grab_buf.frame = frame;
        if(ioctl(grab_fd, VIDIOCMCAPTURE, &grab_buf)<0) {
            perror("VIDIOCMCAPTURE");
            exit(-1);
        }
    }

    frame = 0;
    while(done) {
        i = -1;
        /* waits for the frame sync */

        /* start capture */
        time(&inito);
        while(i < 0) {
            i = ioctl(grab_fd, VIDIOCSYNC, &frame);
```

```
                if(i < 0 && errno == EINTR) {
                     fprintf(stderr,"-");
                     continue;
                     }
                if(i < 0) {
                     perror("VIDIOCSYNC");
                     exit (-1);
                     /* You may want to exit here, because
something has gone
                        pretty badly wrong... */
                }
                break;
           }
           /*finish capture */
           time(&endo);
           fprintf(stderr,"Elapsed time (frame): %d\n", endo-
inito);

           /* refer the frame to save */
           buf = grab_data + grab_vm.offsets[frame];

           /* Process the frame data pointed to by buf */

           save_pnm(buf, x, y, depth);

           /* Once buf is no longer in use by the application...
*/

           grab_buf.frame = frame;
           if(ioctl(grab_fd, VIDIOCMCAPTURE, &grab_buf)<0) {
                perror("VIDIOCMCAPTURE");
                exit(-1);
           }
           /* next frame */
           frame++;
           /* reset to the 1st frame */
           if (frame>=grab_vm.frames) frame = 0;
     }



     close (grab_fd);
}
```

Note that this is a very simple application, but uses the sync‘ing process and mmap. Its a

very clear example on how to do it. Surely, enhancements could be made, regarding data storage and processing, but for the basic understanding it should be enough.

To capture a single frame, this approach can be very simplified. Follows an old VideoDog release exemplifying this.

VIDEODOG version 0.06 - Single Frame capture

```
/* VideoDog - Gleicon S. Moraes
 *
 * Compile with:
 * cc videodog.c -o videodog -O2
 *
 *
 * options you need:
 *
 * h - help
 * x - width
 * y - height
 * w - depth ( in bytes )
 * d - device
 * p - pnm output (24 or 8 bits only )
 *
 * Optional:
 *
 * u - Hue
 * c - Contrast
 * b - Brightness
 * l - Colour
 * s - show device report
 * n - no capture, just setup and exit clean
 *
 * Examples:
 *   ./videodog -s -d /dev/video0
 *              - query the device about settings.
 *   ./videodog -x 320 -y 240 -w 3 -d /dev/video0 -p | cjpeg >
lelel.jpg
 *              - produces a lelel.jpg image with 320x240x24bpp (
3 * 8)
 *   ./videodog -s -d /dev/video0 -u 200
 *              - set Hue == 200 and query the device
 *
 */

#include <stdio.h>
#include <unistd.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <asm/types.h>
#include <linux/videodev.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <asm/types.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>


#define VERSION "0.06"
#define OPTSTR "hx:y:w:d:pu:c:b:l:sni:"
#define DOT fprintf(stderr,".");

void showhelp(void) {
     fprintf (stderr,"Video Dog tool Version %s\n", VERSION);
     fprintf (stderr,"Gleicon S. Moraes\n");
     fprintf (stderr,"Options you really need:\n");
     fprintf (stderr,"-x (number) image width\n");
     fprintf (stderr,"-y (number) image height\n");
     fprintf (stderr,"-w (number) image depth (in bytes) \n");
     fprintf (stderr,"-d device (like /dev/video0 )\n");
     fprintf (stderr,"-p set the output in pnm format (24 or 8
bits only)\n");
     fprintf (stderr, "Other options :\n");
     fprintf (stderr,"-h just show this lines, doesn't help you
ok ?\n");
     fprintf (stderr,"-i input number (0 is the default) \n");
     fprintf (stderr,"-u image Hue\n");
     fprintf (stderr,"-c image Contrast\n");
     fprintf (stderr,"-b image Brightness \n");
     fprintf (stderr,"-l image Colour\n");
     fprintf (stderr,"-s show device report\n");
     fprintf (stderr,"-n Dont capture an image, just setup the
device (Useful if you want to correct something before start
grabbing)\n");
     exit(0);
     }

/* global */

unsigned short int pnm=0;
int x=0, y=0, w=0;
unsigned char *v_device; /* device */

static struct video_picture    grab_pic;
static struct video_capability   grab_cap;
static struct video_channel      grab_vid;
```

```
static struct video_mmap         grab_buf;
static int                       grab_fd, grab_size;
static unsigned char            *grab_data;
int hue=-1, contrast=-1, brightness=-1, colour=-1, nograb=0,
showc=0i, channel=0;

/* prototype */

void swap_rgb24(unsigned char *, int );
unsigned char* grab_one(int *, int *);
int grab_init();


/* signal handler */
void _sighandler (int sig) {
     switch (sig){

          case SIGINT: /* ctrl+x */
               free (v_device);
               close (grab_fd);
               munmap(grab_data, grab_size);
               fprintf (stderr, "Caught SIGINT - Cleaning \n");
               break;
               }
     }
void set_picture() {
     if (hue > -1) grab_pic.hue=hue;
     if (contrast > -1) grab_pic.contrast=contrast;
     if (brightness > -1) grab_pic.brightness=brightness;
     if (colour > -1) grab_pic.colour=colour;
     if (ioctl(grab_fd, VIDIOCSPICT, &grab_pic) == -1) {
          perror ("PICTURE");
          exit (1);
          }
     }

void set_channel() {          /* do nothing if the device doesn't
allow it */
     if (ioctl(grab_fd, VIDIOCGCHAN, &grab_vid) == -1) return;
     grab_vid.channel=channel;
     if (ioctl(grab_fd, VIDIOCSCHAN, &grab_vid) == -1) return;

}

void show_cap()     {    /* mostra settings atuais */
     fprintf (stderr,"\nVideo DOG tool Version %s\n", VERSION);
     fprintf (stderr, "Device Name: %s\n", grab_cap.name);
     fprintf (stderr, "Device: %s\n", v_device);
     fprintf (stderr, "Max Width  : %d\n", grab_cap.maxwidth);
     fprintf (stderr, "Max Height : %d\n", grab_cap.maxheight);
     fprintf (stderr, "Current Settings :\n\n");
```

```
     fprintf (stderr, "\tInput : %i\n", grab_vid.channel);
     fprintf (stderr, "\tBrightness : %i\n",
grab_pic.brightness);
     fprintf (stderr, "\tHue        : %i\n", grab_pic.hue);
     fprintf (stderr, "\tColour     : %i\n", grab_pic.colour);
     fprintf (stderr, "\tContrast   : %i\n", grab_pic.contrast);
     fprintf (stderr, "\n");
     }

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

     int c;
     signal (SIGINT, _sighandler);

     while ((c = getopt (argc, argv, OPTSTR)) != EOF)
          switch (c) {
               case 'x':
                    x=atoi(optarg);
                    break;
               case 'y':
                    y=atoi(optarg);
                    break;
               case 'w':
                    w=atoi(optarg);
                    break;
               case 'p':
                    pnm=1;
                    break;
               case 'd':
                    v_device= (char *)malloc ((strlen(optarg) *
sizeof(char))+1);
                    memset(v_device, 0, strlen(optarg));
                    strncpy(v_device, optarg, strlen(optarg) -1
);

                    break;
               case 'i':
                    channel=atoi(optarg);
                    break;
               case 'u':
                    hue=atoi(optarg);
                    break;
               case 'c':
                    contrast=atoi(optarg);
                    break;
               case 'b':
                    brightness=atoi(optarg);
                    break;
               case 'l':
                    colour=atoi(optarg);
                    break;
```

```
                case 'n':
                     nograb=1;
                     break;

                case 's':
                     showc=1;
                     break;

                case 'h':
                default:
                     showhelp();
                     break;

                }
          if (!x || !y || !w ) nograb=1;
          if (!v_device) {
                fprintf (stderr, " I NEED A DEVICE NAME \n");
                showhelp();
                }

          grab_init();
          set_picture();
          set_channel();
          if (showc) show_cap();

          if (pnm) {
                if (w == 1) {
                     fprintf(stdout, "P5\n%d %d\n255\n", x, y);
                     }
                if (w == 3) {
                     fprintf(stdout, "P6\n%d %d\n255\n", x, y);
                     }
                }
          if (!nograb) fwrite ((unsigned char*)grab_one( &x, &y),
x * y * w, 1, stdout);

          free (v_device);
          close (grab_fd);
          munmap(grab_data, grab_size);

     } /* main */

void swap_rgb24(unsigned char *mem, int n) {
        unsigned char  c;
     unsigned char *p = mem;
     int   i = n ;
     while (--i) {
          c = p[0];
          p[0] = p[2];
          p[2] = c;
          p += 3;
```

```
          }
     }

int grab_init() {
        if ((grab_fd = open(v_device,O_RDWR)) == -1 ) {
          perror("open videodev");
          exit(1);
          }
     if (ioctl(grab_fd,VIDIOCGCAP,&grab_cap) == -1) {
             fprintf(stderr,"wrong device\n");
          exit(1);
          }
     memset (&grab_pic, 0, sizeof(struct video_picture));
     if (ioctl (grab_fd, VIDIOCGPICT, &grab_pic) == -1) {
          fprintf (stderr, "no pict");
          exit(1);
          }

     switch (w) {
          case 1:
               grab_buf.format = VIDEO_PALETTE_GREY;
               break;
          case 2:
               grab_buf.format = VIDEO_PALETTE_RGB565;
               break;
          case 3:
          default:
               grab_buf.format = VIDEO_PALETTE_RGB24;
               break;
          }

     grab_buf.frame  = 0;
     grab_buf.width  = x;
     grab_buf.height = y;
     grab_size = x * y * w;
     grab_data =
mmap(0,grab_size,PROT_READ|PROT_WRITE,MAP_SHARED,grab_fd,0);
     return(1);
     }

unsigned char* grab_one(int *width, int *height) {
     for (;;) {
          if (-1 == ioctl(grab_fd,VIDIOCMCAPTURE,&grab_buf)) {
               perror("ioctl VIDIOCMCAPTURE");
               } else {
                    if (-1 ==
ioctl(grab_fd,VIDIOCSYNC,&grab_buf)) {
                         perror("ioctl VIDIOCSYNC");
                    } else {
                         //DOT
                         fprintf(stderr,"len: %d\n", grab_size);
```

```
                        if (w == 3) swap_rgb24(grab_data,
grab_buf.width * grab_buf.height);
                        *width  = grab_buf.width;
                        *height = grab_buf.height;
                        return grab_data;
                    }
                }

            }
        }
```

Besides lots of bugs, none of them too critical, this application shows how to open, get a frame and close the device. It serves well for educational purposes.

FINAL THOUGHTS AND COMMENTS

Research through video formats are needed, to suit better your needs. I‘ve been using RGB24, because its easier to work with, and matches with my other applications. Note that the driver gives you a BGR24 frame, and you need to swap it before using. In RGB8 or other colorspaces it is not true.

LINKS AND REFERENCES

http://videodog.cjb.net - Videodog Home (my home page)
http://bytesex.org/xawtv/ - XawTV source code, is the canonical application to v4l development.
http://www.glue.umd.edu/~ankurm/video4linux/introduction.html - Another v4l Intro
https://listman.redhat.com/mailman/listinfo/video4linux-list - v4l maling list archives
http://www.exploits.org/v4l/ - Resource list, from API to apps.

Many concepts used here were kindly teached by other people and seen applied in codes overthere. Research is fundamental and the key to a working projetct. Do yours !