USB-4 Captura de Imagens/Vídeo de Webcam USB

Meios Eletrônicos I 2011

Objetivos de Aprendizado

- Como capturar imagens/vídeos de Webcams USB em ambiente LINUX

LINUX USB

- Bibliotecas e módulos
  - V4l2
  - Libv4l – prove a compatibilidade com o V4L1 entre outros
    - LD_PRELOAD=/usr/lib64/libv4l/v4l1compat.so (aplicacoes v4l1 possam rodar sobre dispositivos v4l2)
    - LD_PRELOAD=/usr/lib64/libv4l/v4l2convert.so (conversao de formato)
  - Libwebcam (do grupo de desenvolvimento quickcam)

Linux USB

- Webcam USB geralmente montado em:
  - /dev/video0
- Controle/Status/configuração a partir da chamada ioctl()
- Drivers:
  - Específicos de fabricante/chip:
    - PWC (philips)
  - UVC: USB Video Class
  - GSPCA

Linux USB

- Utilitários importantes:
  - Lsusb: descobrir o vendor:product ID da webcam USB
  - Lsmod: descobrir qual foi (eventualmente) o módulo driver carregado
  - Dmesg: descobrir qual foi o hardware descoberto pelo Linux e eventualmente como foi “montado”
  - USBVIEW e USBTREE:
    - Se a distribuição Linux montar o USBFS (ou UBDEVFS), permite descobrir a arvore de dispositivos USB e seus parâmetros
    - /proc/bus/usb/devices
  - Uvcdynctrl: permite mostrar as propriedades da(s) webcam(s) instaladas

Aplicações de Vídeo no LINUX

- Cheese
- Xawtv
  - streamer
- MPLAYER
- V4LControl

Programação V4L

Programas exemplo V4L1

- http://alumnos.elo.utfsm.cl/~yanez/video-for-linux-sample-programs/

Programas Exemplo V4L2

- http://alumnos.elo.utfsm.cl/~yanez/video-for-linux-2-sample-programs/
  - Capturer_mmap
  - Capturer_read
  - viewer

Executandoo capturer_mmap:

- capturer_mmap
- To get usage info from command line, run:
- # ./capturer_mmap -h
- To get info about the configuration options of a certain device, run:
- # ./capturer_mmap [-d Capture Device]
- Capture Device: Video For Linux II device, eg: /dev/video
- To capture video, and throw the data to standard output, run:
- # ./capturer_mmap [-D Capture Device] [-i Input] [-s Standard] [-w Window Size] [-p Pixel Format]
- Capture Device: Video For Linux II device, eg: /dev/video (MUDAR PARA /dev/video0!!!!!)
- Input: Input channel of the capture device, if you want to know the available values for a certain device, just use the program with the -d switch
- Standard: A number, who represents the TV norm used (eg: NTSC). if you want to know the values for a certain device, use the -d switch
- Window Size: Could be: 320*240 or 640*480
- Pixel Format: A number, who represents the format used to save the images in the memory buffer, the values could be:
- 0 YUV420
- 1 RGB565
- 2 RGB32

Executando o Viewer:

- To get info of program usage, run:
- # ./viewer -h

- To display video, receiving data from the standard input, run:
- #./viewer [-w Window Size] [-p Pixel Format]

- Window Size: Could be: 320*240 or 640*480

- Pixel Format: A number, who represents the format used in the memory buffer
- 0 YUV420
- 1 RGB565
- 2 RGB32

Tamanhos

sQCIF	128x96
QSIF	160x120
QCIF	176x144
SIF	320x 240
CIF	352x288
VGA	640x48

Formatos

- YUV
- RGB

Compilando:

- Instalar o “xlibs development files”. Exemplo, para o Debian:
  - # apt-get install xlibs-dev
- Descompactando e Compilando:
  - # tar -zxvf V4l2_samples-0.4.tar.gz
  - # cd V4l2_samples-0.4
  - # make

Executando: mais detalhes

- To capture video with a resolution of 640x480, with a webcam who uses pwc driver (Phillips, Logitech), just run:
- # modprobe -r pwc
- # modprobe -v pwc fps=15 compression=3 mbufs=4 fbufs=4 size=vga
- # ./capturer_mmap -D /dev/video -w 640*480 -p 0 | ./viewer -w 640*480 -p 0

- To capture video with a resolution of 320x240, with a webcam who use pwc driver (Phillips, Logitech), just run:
- # modprobe -r pwc
- # modprobe -v pwc fps=30 compression=3 mbufs=4 fbufs=4 size=sif
- # ./capturer_mmap -D /dev/video -w 320*240 -p 0 | ./viewer -w 320*240 -p 0

Input/Output Method Negotiation

- To capture video with a resolution of 320x240, with a capture card who uses bttv driver (i.e. TV Master+FM), just run:
- # modprobe -r bt878
- # modprobe -r bttv
- # modprobe modprobe bttv tuner=17 card=70 radio=1
- # ./capturer_mmap -D /dev/video0 -w 320*240 -p 0 | ./viewer -w 320*240 -p 0

Programas V4L2

Estrutura:

- Open the device.
- Properties Negotiation (video input, video standard, and more)
- Pixel Format Negotiation
- Input/Output Method
- Main Loop
- Close the device

As aplicações V4L2 devem ser programadas usando a função ioctl:

- ioctl(<device_descriptor>, <operation_code>, <pointer_to_structure>)

op. code	I/O	structure
VIDIOC_QUERYCAP	IOR	struct v4l2_capability
VIDIOC_RESERVED	IO	1
VIDIOC_ENUM_FMT	IOWR	struct v4l2_fmtdesc
VIDIOC_G_FMT	IOWR	struct v4l2_format
VIDIOC_S_FMT	IOWR	struct v4l2_format
VIDIOC_G_COMP	IOR	struct v4l2_compression
VIDIOC_S_COMP	IOW	struct v4l2_compression
VIDIOC_REQBUFS	IOWR	struct v4l2_requestbuffers
VIDIOC_QUERYBUF	IOWR	struct v4l2_buffer
VIDIOC_G_FBUF	IOR	struct v4l2_framebuffer
VIDIOC_S_FBUF	IOW	struct v4l2_framebuffer
VIDIOC_OVERLAY	IOWR	int
VIDIOC_QBUF	IOWR	struct v4l2_buffer
VIDIOC_DQBUF	IOWR	struct v4l2_buffer
VIDIOC_STREAMON	IOW	int
VIDIOC_STREAMOFF	IOW	int
VIDIOC_G_PARM	IOWR	struct v4l2_streamparm
VIDIOC_S_PARM	IOW	struct v4l2_streamparm
VIDIOC_G_STD	IOR	v4l2_std_id
VIDIOC_S_STD	IOW	v4l2_std_id
VIDIOC_ENUMSTD	IOWR	struct v4l2_standard
VIDIOC_ENUMINPUT	IOWR	struct v4l2_input
VIDIOC_G_CTRL	IOWR	struct v4l2_control
VIDIOC_S_CTRL	IOW	struct v4l2_control
VIDIOC_G_TUNER	IOWR	struct v4l2_tuner
VIDIOC_S_TUNER	IOW	struct v4l2_tuner
VIDIOC_G_AUDIO	IOWR	struct v4l2_audio
VIDIOC_S_AUDIO	IOW	struct v4l2_audio
VIDIOC_QUERYCTRL	IOWR	struct v4l2_queryctrl
VIDIOC_QUERYMENU	IOWR	struct v4l2_querymenu
VIDIOC_G_INPUT	IOR	int
VIDIOC_S_INPUT	IOWR	int
VIDIOC_G_OUTPUT	IOR	int
VIDIOC_S_OUTPUT	IOWR	int
VIDIOC_ENUMOUTPUT	IOWR	struct v4l2_output
VIDIOC_G_AUDOUT	IOWR	struct v4l2_audioout
VIDIOC_S_AUDOUT	IOW	struct v4l2_audioout
VIDIOC_G_MODULATOR	IOWR	struct v4l2_modulator
VIDIOC_S_MODULATOR	IOW	struct v4l2_modulator
VIDIOC_G_FREQUENCY	IOWR	struct v4l2_frequency
VIDIOC_S_FREQUENCY	IOW	struct v4l2_frequency
VIDIOC_CROPCAP	IOR	struct v4l2_cropcap
VIDIOC_G_CROP	IOWR	struct v4l2_crop
VIDIOC_S_CROP	IOW	struct v4l2_crop
VIDIOC_G_JPEGCOMP	IOR	struct v4l2_jpegcompression
VIDIOC_S_JPEGCOMP	IOW	struct v4l2_jpegcompression

Open

Device Name		Minor Number	Description
From	To		
/dev/video0	/dev/video63	0-63	Video Capturer Devices
/dev/radio0	/dev/radio63	64-127	AM/FM Radio Devices
/dev/vtx0	/dev/vtx31	192-223	Teletext Devices
/dev/vbi0	/dev/vbi15	224-239	VBI Devices

Property Negotiation

- Video Input.
- Video Output.
- Norm (only for input devices).
- Modulator (only for output devices).
- Input Channel.
- Window Size.

Pixel Format Negotiation

- Pixel format negotiation is also a part of the properties negotiation, but is on a separated section, just to be explained more in detail. The pixel format is how every pixel is stored in memory, and the application need to know this format to allow the properly interpretation of that pixel. There are two "families" of pixel formats RGB and YUV.

- YUV format are useful in the input of video codecs, in fact YUV formats have a basic kind of compression, because they reduce the crominance components, because these are less perceptible to the human eye. On the other side RGB format keep all the crominance information, and these are useful sometimes for video processing or displaying on a window.

- In the most cases the devices capture natively in YUV formats, and in these programs, the video is converted to RGB formats, for displaying in the viewer, this takes some time of cpu

Input/Output Method Negotiation

- READ/WRITE functions
  - The devices supports this method, when the flag V4L2_CAP_READWRITE is set on the capabilities member of the structure V4l2_capability. The use of this function could be slower than the other methods, because all the data must be copied.
- Memory mapping
  - This is the fastest method, read and write functions are not needed, instead of those, the mmap() function is used. This functions returns a pointer to the start of a valid memory area, this memory is used by the application to read the data. A device support this method when the flag V4L2_CAP_STREAMING of the capabilities field in the v4l2_capability struct is set.
- User Pointer
  - A device supports this method if the field V4L2_CAP_STREAMING in the member capabilities of the struct V4l2_capability returned by the VIDIOC_QUERYCAP ioctl is set. If the particular user pointer method (not only memory mapping) is supported must be determined by calling the VIDIOC_REQBUFS ioctl.
  - This method combines the advantage of the both previous methods. In the user pointer method buffers are allocated by the application and can be shared memory (mmap) or virtual. Only pointers to data are exchanged, these pointers and meta-information are passed in struct v4l2_buffer. The driver must be switched into user pointer I/O mode by calling the VIDIOC_REQBUFS with the desired buffer type. This allow the use of DMA.

Main Loop

- If user pointer or memory mapping is used, that are streaming oriented methods, the first step is the start of the transmission of data.
- In addition these methods have buffers with queues, in that way, V4L2 enqueue data, and the application dequeue the data, with a FIFO criteria.
- In the case of output devices, the process is the same but inverse. In addition the buffer must be previously configured with the properties of the frames, to know the size of one frame, for allocation of memory. And at the end of the loop, the data transmission must be stopped and the buffers freed.
- In the case of use of read/write functions in the main loop, the application only have one memory buffer, that one allows the store of one frame. On a capture application in every loop the read function must be called to capture one frame.
- For the 3 methods is valid the use of the select() function, to wait for events, avoiding the use of CPU when is waiting.

Close the device

- NÃO SE ESQUEÇA DE FECHAR O DISPOSITIVO!!!!