최종연구보고서

리눅스 디지틀 비디오 레코더

수탁기관 상명대학교

2004년 7월 31일

한국소프트웨어진흥원

제 출 문

한국소프트웨어진흥원장 귀하

본 보고서를 "공개 S/W 프로젝트 활용 S/W 전문인력 양성 사업" 중 "리눅스 디지틀 비디오 레코더" 과제(연구 기간: 2003년 10월 1일 ~ 2004년 7월 31일)의 최종연구보고서로 제출합니다.

2004년 7월 31일

수 탁 기관: 상명대학교

수탁기관장: 서명덕 총장 (인)

연구책임자: 신동하 교수

참여 교수: 백윤철 교수

참여 학생: 김인영, 이우철, 황병주, 유홍기, 정성욱, 정훈

목 차

제1장 개요

제1장은 본 보고서의 개요로서 연구의 목적, 연구 범위, 추진 방법, 인력 양성 결과, 연구 결과물, 기대 효과 및 향후 계획에 대하여 순서적으로 기술한다.

제1절 연구 목적

본 연구의 최종 목적은 "공개 S/W 프로젝트 활용 S/W 전문인력 양성"이다. 여기서 "공개 소프트웨어 프로젝트 활용"의 의미는 크게 두가지이다. 첫째는 프로젝트를 진행하는 과정에 기 공개된 소프트웨어 기술을 사용한다는 의미이고 두번째는 본 연구에서 개발하는 소프트웨어 및 기술 문서를 공개한다는 의미이다.

1. 개발 주제 선정

본 연구에서는 위에서 설명한 연구 목적을 달성하기 위하여 "리눅스 디지털 비디오 레코더" 소프트웨어를 개발한다. 개발하는 디지털 비디오 레코더(DVR) 소프트웨어는 아래과 같은 이유 때문에 본 연구의 목적을 최대로 달성할 수 있을 것이라고 판단하였다.

- 흥미를 끄는 소프트웨어: 개발하는 소프트웨어는 비디오, 오디오 등과 같이 개발자에게 흥미를 유발하는 요소를 많이 가지고 있다. 이 점은 개발 과정에서 개발자가 계속적으로 흥미를 느낄 수 있게 하기 때문에 인력 양성에 매우 효과적인 요소로 작용한다.

- 다양한 기술의 분포: 개발하는 소프트웨어는 소프트웨어를 계층(software layer)으로 볼 때 아래 계층에서부터 위 계층에 이르는 다양한 기술적 사항의 이해를 필요로 한다. 예를 들어 디바이스 드라이버의 이해, 여러 라이브러리의 이해, 응용 소프트웨어 개발의 이해 및 사용자 인터페이스의 이해 등의 기술을 요구한다. 결과적으로 본 과제를 참여하는 학생들은 다양한 기술적 사항을 습득하게 된다.

- 적절한 난이도: 본 과제에서는 약 10개월 동안 완전한 소프트웨어를 완성하여야 한다. 이를 위하여 연구 주제는 학생들이 기간 내에 개발 가능한 난이도를 가지고 있어야 한다. 본 연구가 요구하는 기술은 C 프로그래밍 기술, 리눅스 시스템 호출 인터페이스 기술, 다양한 디바이스 입출력 기술, MPEG 인코딩/디코딩 기술, 모션 검출 기술, 웹 스트림 기술 등이다. 이 모든 기술에 대한 이론은 학생들이 3 ~ 4학년 과정에서 대부분 배운 내용이기 때문에 적절한 난이도라고 판단된다.

- 결과물의 활용도: 리눅스는 윈도우 운영체제제 비하여 상대적으로 소수의 미디어 프로그램

을 제공하고 있다. 많은 리눅스 사용자들이 이 점에 대하여 많은 아쉬움을 표현하고 있다. 본 연구에서 개발하는 소프트웨어는 이점에서 많은 기여를 하리라고 판단되어서 개발 주제를 선정하였다.

2. 사용 공개 소프트웨어 기술

본 연구를 수행하기 위해서는 아래 표에서와 같은 다양한 공개 소프트웨어 기술을 사용한다. 이들 기술은 소스 코드와 함께 자유롭게 제공되고 있다.

<표 1.1> 사용 공개 소프트웨어 기술

사용 기술	사용 분야	자료 URL
video4linux	video capture	http://linux.bytesex.org/v4l2/bttv.html
OSS	audio capture	http://www.opensound.com/oss.html
ffmpeg	video/audio record/play	http://ffmpeg.sourceforge.net
Jpeg library	video stream	http://www.ijg.org
Xfree86	video display,	http://www.xfree86.org

제2절 연구 범위

본 절에서는 개발할 소프웨어의 기능적 범위 및 기술적 범위를 기술하고 개발할 소프트웨어의 구조 및 구성에 대하여 설명한다..

1. 기능적 범위

- TV 카드 입출력 기능: TV 카드에서 들어오는 비디오 및 오디오 데이터를 입력받아 컴퓨터 화면과 스피커를 통하여 출력하는 기능.

- TV 카드 동영상 저장 기능: TV 카드에서 들어오는 비디오 및 오디오 데이터를 비디오 및 오디오 인코딩 기술을 사용하여 인코딩한 다음 디스크에 파일로 기록하는 기능.

- 저장된 동영상 파일의 재생 기능: 디스크에 파일로 저장된 TV 프로그램을 화면과 스피커를 통하여 재생하는 기능.

- 카메라 동영상 스트림 기능: USB 카메라에서 들어오는 비디오 데이터를 인터넷을 통하여 스트림하여 원격에서 웹 브라우저 상에서 볼수 있게 하는 기능.

- 모션 검출 기능: USB 카메라에서 들어오는 비디오 입력에서 묘션 변화 부분 만을 검출하여 디스크에 파일로 저장하는 기능.

- 저장된 동영상 웹 인터페이스 기능: USB 카메라에서 입력 받은 화면과 검출된 화면은 동영상 인코딩 되어 파일로 저장된다. 이들 파일은 모두 원격에서 웹 브라우저 상에서 재생해 볼 수 있다.

2. 기술적 범위

- 본 연구에서 사용하는 TV 카드 및 USB 카메라의 디바이스 드라이버 프로그램은 본 연구에서 개발하는 소프트에어가 아니다. 본 연구에서는 리눅스 운영체제에서 가장 안정적으로 동작되는 bttv 및 pwc 디바이스 드라이버를 사용한다.

- 본연구에서는 오디오 신호의 재생을 위하여 사운드 카드를 사용한다. 이때 ALSA 사운드 드라이버를 사용한다.

- 본 연구의 비디오 스트림 서버에 사용되는 정지 화면의 JPEG 인코딩 방법의구현을 위해서는 공개 소스 JPEG 라이브러리를 사용한다.

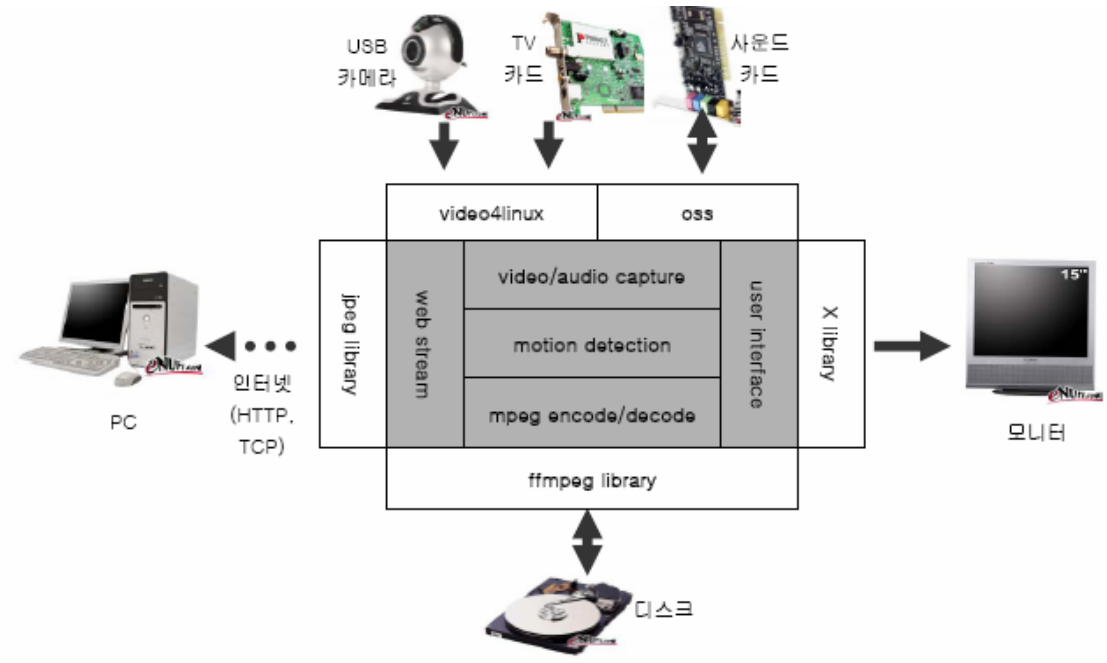
- 본 연구에서는 비디오 및 오디오 신호를 저장할 때 MPEG1 및 MP2 인코딩 방법을 사용하여 인코딩한 후 저장한다. 인코딩을 위한 소프트웨어는 본 연구에서 개발하지 않고 공개 소스 소프트웨어인 ffmpeg 라이브러리를 사용한다.

- 동영상을 화면에 디스플레이하기 위하여 본 연구에서는 XFree86 라이브러리를 사용한다. 본 라이브러리는 본 연구에서의 개발 대상은 아니다.

- 본연구에서는 비디오 및 오디오 신호의 입력, 출력, 디스크로의 저장, 인터넷으로의 스트림 등을 직접 C 언어를 사용하여 리눅스 운영체제에서 프로그램한다. 즉 응용 프로그램은 모두 직접 C 언어로 프로그래밍 한다.

3. 소프트웨어의 구조

본 연구에서 개발하는 리눅스 디지틀 비디오 레코드의 계략적인 구조는 다음 그림과 같다. 이 그림은 사용하는 디바이스(작은 그림), 이용하는 디바이스 드라이버 및 라이브러리(투명 네모) 및 본 연구에서 직접 프로그래밍하는 프로그램(회색 네모)을 구별하여 나타내고 있다.



<그림 1.1> 리눅스 디지틀 비디오 레코더의 구조

4. 소프트웨어 구성

본 연구에서 개발하는 소프트웨어는 용도에 따라 크게 2가지로 나누어지는데 카메라 부분을 다루는 소프트웨어를 “DVR Motion”이라고 명명하고 TV 부분을 다루는 소프트웨어를 “DVR TV”라고 명명한다. 이들 각각의 소프트웨어는 독립적으로 수행한다.

<표 1.2> 개발 소프트웨어 구성

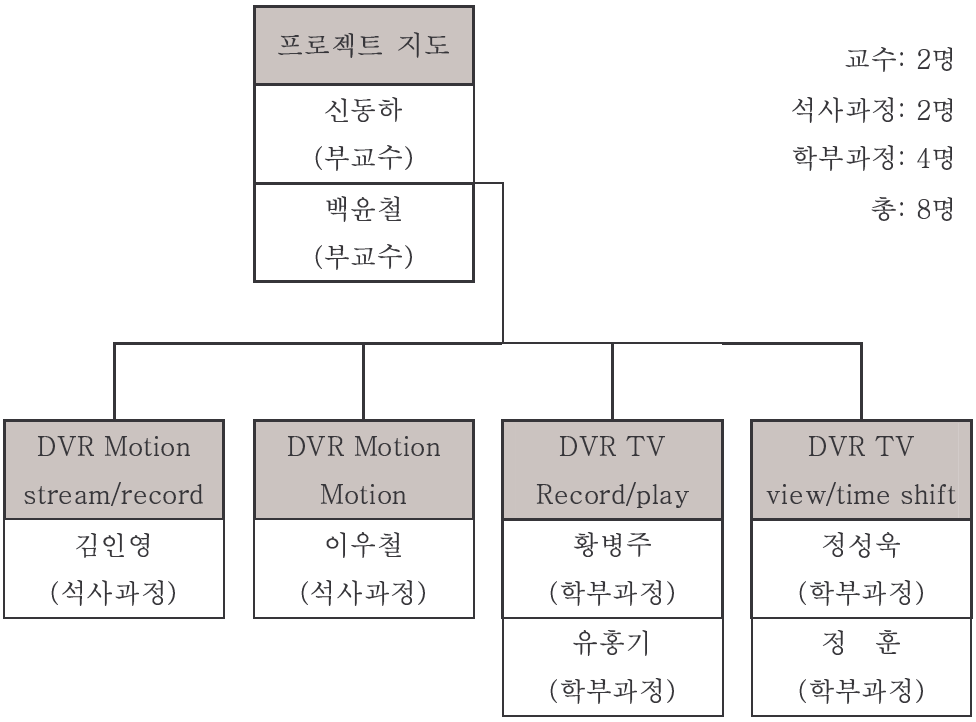
개발 소프트웨어	사용 디바이스	주요 기능
DVR Motion	USB 기반 카메라 (예: Logitech QuickCam 4000 Pro 등)	카메라 동영상의 motion 검출, 실시간 인터넷 stream, record
DVR TV	PCI 기반 TV 카드 (예: Pinnacle PCTV 등) 일반 사운드 카드	TV 동영상 및 사운드의실시간 view, time shift, record 및 play

제3절 추진 방법

본 절에서는 본 연구를 수행하는 개발 팀의 구성과 개발 방법론에 대하여 기술한다.

1, 개발 팀 구성

본 연구를 수행하는 개발팀은 상명대학교 소프트웨어학부의 리눅스 연구회인 "ELF(ELF is a Linux Family)"에서 활동하는 교수 2명 및 학생 6명으로 구성된다. 학생 6명 중 석사 과정 학생 2명은 DVR Motion 부분을 개발하고 학부 학생 4명은 DVR TV 부분을 구현한다. (참고: DVR Motion 부분이 프로그램의 양은 작으나 모션 검출이라는 기술적 요소가 포함되어 있어서 석사 학생에게 적합함)



<그림 1.2> 개발 팀 구성

2. 개발 방법

본 연구에서 사용하는 개발 방법론은 전통적인 소프트웨어 개발 방법론인 "Waterfall 방식"을 기반으로 진행하되 많은 기술적 사항을 적시에 프로그래밍으로 확인해 보기 위하여 각 개발 단계에서는 부분적인 "Prototyping 방식"을 활용한다. 각 개발 단계별 작성 문서 및 프로그래밍 결과물은 다음 표와 같다.

<표 1.3> 개발 단계 및 결과물

개발 단계	문서 결과물	프로그래밍 결과물
요구 분석 단계 2004년 1월 1일 ~ 2004년 2월 28일 (7주)	요구분석서, 기술문서, 프로그램	기술 습득을 위한 프로그램 (video/audio input, mpeg record/play)
설계 단계 2004년 3월 1일 ~ 2004년 5월 16일 (11주)	설계서, 기술문서, 프로그램	기술 향상을 위한 프로그램 (video stream, motion detection, time shift)
구현 단계 2004년 5월 17일 ~ 2004년 7월 31일 (11주)	구현서, 기술문서, 프로그램	구현한 프로그램 (DVR Motion, DVR TV)

제4절 인력 양성 결과

본 연구의 목적은 소프트웨어 개발 인력 양성이다. 본 절에서는 인력 양성 결과를 6명의 인력에 대한 "기술적 분야"와 "기술적 성숙도"로 분류하여 기술하였다.

1. 기술적 분야

본 과제에서 양성된 소프트웨어 개발 인력의 수는 총 6명이다. 이중 석사 과정 학생이 2명이고 학부 학생이 4명이다. 아래는 각 개발 인력이 습득한 기술적 분야를 나타낸 표이다.

<표 1.4> 각 인력의 기술적 분야

인력	과정	기술적 분야
김인영	석사	Video4linux 기술, MJPEG 스트림 기술, MPEG 저장 기술
이우철	석사	Video4linux 기술, 모션 검출 기능, 프로젝트 관리 기술
황병주	학부	Video4linux 기술, OSS 프로그래밍 기술, MP2 인코딩 기술
유홍기	학부	Video4linux 기술, MPEG 인코딩 기술, MPEG 디코딩 기술
정성욱	학부	Video4linux 기술, MPEG 인코딩 및 프로세스간 통신 기술
정 훈	학부	Video4linux 기술, X 윈도우 프로그래밍 기술, GUI 기술

2. 기술적 성숙도

각 개발자 별로 개발 참여 초기인 2004년 1월 1일과 최종 보고서 작성 시점인 2004년 7월 31일의 기술적 성숙도를 비교하여 표시하면 다음 표와 같다. 여기서 점수의 의미는 다음과 같다.

1점 - 기술적 이해의 최하 단계
2점 - 기술의 이론을 이해하기 시작하는 단계 (프로그램을 할 수 없는 단계)
3점 - 기술적 이론을 이해하고 실제 프로그래밍을 시작할 수 있는 단계
4점 - 프로그래밍을 성숙하게 할 수 있는 단계
5점 - 고급 프로그래밍을 할 수 있는 단계

<표 1.5> 개발자 별 질적 성숙도 변화표

인력	과정	요구 분석 단계					설계 단계					구현 단계				
		1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
이우철	석사															
김인영	석사															
황병주	학사															
유홍기	학사															
정성욱	학사															
정 훈	학사															

제5절 연구 결과물

본 연구 수행 중 작성한 문서(기술 및 비기술 문서 포함) 및 소스 코드는 본 프로젝트의 홈 페이지인 http://pl.smu.ac.kr/researches/projects/linux-dvr에 공개되어 있다. 또한 프로젝트의 각 단계별 결과물(기술 문서 및 소스 코드)은 http://www.oss.or.kr/linuxdvr에 공개되어 있다. 과제 종료 시점인 2004년 7월 31일 현재 결과물 총 25종으로 아래 표와 같다.

<표 1.6> 연구 결과물

문서 및 소스 이름	소스 포함
1. video4linux API 스펙	O
2. video4linux를 사용하여 비디오 캡쳐하기	O
3. MythTV 설치하기	
4. PNM 포맷 이해하기	O
5. RedHat 9에 vloopback 설치 방법	
6. video4linux를 사용한 실시간 디스플레이	O
7. X 라이브러리를 사용한 실시간 디스플레이	O
8. OSS 오디오 프로그래밍	O
9. 모션 검출 알고리즘	
10. DVR 기능 비교	
11. Linux DVR 요구 명세서	
12. X 라이브러리와 LessTif을 사용한 디스플레이 방법	O
13. ffmpeg을 사용한 mpeg 디코딩 방법	O
14. OSS와 ffmpeg을 사용한 mp2 오디오 레코딩 방법	O
15. ffmpeg을 사용한 mpeg 인코딩 방법	O
16. Linux DVR TV 설계서	
17. Linux DVR Motion 설계서	
18. ffplay 분석서	
19. video4linux 속도 향상을 위한 double buffering 방법	
20. bt848 및 bt878 칩셋이 장착된 TV 카드 시험	
21. 세가지 모션 검출 알고리즘	
22. Linux DVR TV 구현서	
23. Linux DVR Motion 구현서	
24. Linux DVR TV 소스 코드	O
25. Linux DVR Motion 소스 코드	O

위 결과물은 본 보고서의 2장 ~ 5장에서 자세하게 설명하였다. 2장 사전 기술 조사 단계에서는 video4linux API 스팩, video4linux를 사용하여 비디오 캡쳐하기, video4linux를 이용한 실시간 디스플레이, X 라이브러리를 사용한 실시간 디스플레이, OSS 오디오 프로그래밍, 그리고 기존 DVR 기능 비교로 총 6가지 기술 문서를 기술하였다. 3장 요구 분석 단계에서는 리눅스 DVR 요구 명세서를 기술하였다. 4장 설계 단계에서는 DVR TV와 DVR Motion의 설계서 뿐만 아니라, X 라이브러리와 LessTif을 사용한 디스플레이 방법, ffmpeg을 사용한 mpeg 인코딩 방법, ffmpeg을 사용한 mpeg 디코딩 방법, 그리고 OSS 및 ffmpeg을 사용한 mp2 오디오 레코딩 방법 등을 기술하였다. 5장 구현 단계에서는 DVR TV와 DVR Motion 구현서 뿐만 아니라, ffplay 분석서, video4linux 속도 향상을 위한 double buffering 방법, bt848 및 bt878 칩셋이 장착된 TV 카드 시험, 그리고 세가지 모션 검출 알고리즘 등을 기술하였다.

제6절 기대 효과

본 연구 수행의 기대 효과를 프로그래머 인력 양성 측면, 개발한 소프트웨어의 이용 측면, 연구 개발 환경 구축 측면으로 나누어 설명한다.

1. 프로그래머 인력 양성

본 연구에서는 리눅스 운영체제 상에서 비디오 및 오디오 전문 프로그래머를 양성하였다. 개발 초기에 각 개발자는 C 프로그래밍 및 리눅스 시스템 호출 프로그래밍 기술 만을 가지고 있는 학생이였지만 연구 수행 종료 시점에는 리눅스 비디오 및 오디오 전문 프로그래머로 양성되었다. 리눅스 운영체제 상에서 이 분야의 전문 인력이 부족한 점을 고려한다면 앞으로 이들이 실무에 투입되어 전문 지식과 경험을 충분히 발휘할 것으로 기대된다.

2. 개발한 소프트웨어의 이용

개발한 소프트웨어는 현재 리눅스 상에서 비디오 관련 프로그램이 부족한 점을 고려하면 매우 좋은 결과라고 할 수 있다. 개발한 소프트웨어는 앞으로 여러 리눅스 배포판에 삽입되어 여러 사용자가 이용하기를 기대한다. 일차적으로 리눅스 데스크탑에서 사용가능하지만 이차적으로 임베디드 시스템으로의 개발도 충분히 가능하다.

3. 연구 개발 환경 구축

본 연구 수행의 큰 결과라면 학교 내에 리눅스 전문 프로그래밍을 할 수 있는 인력이 구축되었다는 점이다. 이들이 곧 졸업하여 사회로 배출되겠지만 이들의 후배들이 이들이 개발한 소프트웨어 개발 기술을 바탕으로 앞으로는 더욱더 기술적 수준이 높은 소프트웨어의 개발도 충분히 가능하리라고 기대한다.

제2장 사전 기술 조사 단계

본 장에서는 "리눅스 디지틀 비디오 레코더" 프로젝트를 수행하기 위해 필요한 사전 기술들을 소개한다. 사전 기술로는 video4linux API 스팩, video4linux를 사용하여 비디오 캡쳐하기, MythTV 설치하기, PNM 포맷 이해하기, RedHat 9에 vloopback 설치하기, video4linux를 이용한 실시간 디스플레이, X 라이브러리를 사용한 실시간 디스플레이 OSS 오디오 프로그래밍, 모션 검출 알고리즘, 기존 DVR 기능 비교 등이 있다. 사전 기술에 대한 문서들은 OSS 프로젝트에 공개되어 있으며, 본 장에서는 video4linux API 스팩, video4linux를 사용하여 비디오 캡쳐하기, video4linux를 이용한 실시간 디스플레이, X 라이브러리를 사용한 실시간 디스플레이 OSS 오디오 프로그래밍, 그리고 DVR 기능 비교에 대해 기술한다.

제1절 video4linux API 스팩

본 절에서는 video4linux를 이용하여 영상을 캡쳐하기 위해 필요한 API를 정리, 요약 하고 있다. 본 문서에서는 video4linux의 간단한 정의와 video4linux 디바이스의 정보를 얻고 조작하는데 필요한 Capability Query, Video Source, Capture Windows, Image Properties, Reading Images에 관련된 API를 다룬다.

1. video4linux

video4linux는 리눅스에서 비디오 디바이스를 제어하고 사용하기 위한 API를 말한다. 비디오 디바이스는 튜너를 포함한 비디오 카드나 웹캠과 같이 영상 정보를 입력할 수 있는 장치를 뜻하며, 튜너를 포함한 디바이스의 경우 TV나 AM/FM 라디오, Teletext등의 방송을 시청/청취할 수 있다.

video4linux는 리눅스 커널에 기본적으로 포함되어 있으며, 커널 컴파일을 통해 커널 모듈, 혹은 정적으로 컴파일하여 사용이 가능하다.

video4linux는 각 디바이스를 위해 다음의 파일을 제공하고 있으며, video4linux를 이용하기 위해서는 다음의 디바이스 파일을 Open하고 검색하여 지원하는 기능을 찾아낸 후 API를 이용하여 사용할 수 있다.

디바이스 이름	마이너 번호	기능
/dev/video	0-63	비디오 캡쳐 인터페이스
/dev/radio	64-127	AM/FM 라디오 디바이스

/dev/vtx	192-223	Teletext 인터페이스 칩
/dev/vbi	224-239	Raw VBI 데이타 (Intercast/teletext)

현재는 video4linux보다 낳은 기능과 다양한 디바이스의 지원을 위한 Video For Linux Two가 등장하였다.

* Teletext는 문자방송을 뜻한다.
* VBI는 Vertical Blanking Interval의 약자로 화면에 보이지 않지만, 이 부분을 이용하여 Digital Data가 방송 되는 것을 뜻한다.
* Intercast는 VBI를 이용하여 TV와 Internet을 융합한 시스템을 말한다.

2. Capability Query

어느 특정 디바이스를 사용하고자 한다면, 디바이스의 특징을 정확히 알고 있겠지만, 대부분의 프로그램은 프로그램의 사용 환경에 따라 다른 디바이스를 제어하게 된다. 따라서, 디바이스를 사용하기 전에 해당 디바이스의 기능을 정확히 알아야 할 필요가 있다. 이러한 경우 다음의 ioctl Call을 사용하게 된다.

```
#include <errno.h>
#include <sys/ioctl.h>
#include <linux/videodev.h>

int ioctl(int filedes, VIDIOCGCAP, struct video_capability *);
```

VIDIOCGCAP ioctl Call은 디바이스 파일의 파일 디스크립터(File Descripter)를 인자로 받아 해당 디바이스의 기능 정보를 video_capability라는 미리 정의된 구조체에 저장한 후 성공 하였을 경우 0을 반환하게 된다. 만일 기능 정보를 얻어오는데 실패하였을 경우 errno.h 파일에 정의 된 에러 코드를 음수로 바꾼 값이 반환된다. 따라서 반환 값(Return value)가 0보다 작다면, 실패를 뜻한다.

성공한 후 반환되는 struct video_capability의 정의는 다음과 같다.

자료형	이름	의미
char [32]	name	인터페이스의 일반적인 이름
int	type	인터페이스의 타입
int	channels	라디오/TV의 입력 채널의 수

int	audios	오디오 디바이스의 수
int	maxwidth	캡쳐의 최대 가로 길이(단위: 픽셀)
int	maxheight	캡쳐의 최대 세로 길이(단위: 픽셀)
int	minwidth	캡쳐의 최소 가로 길이(단위: 픽셀)
int	minheight	캡쳐의 최소 세로 길이(단위: 픽셀)

위의 값들 중 channels와 audios는 유효한 자원이 존재하지 않을 경우 0의 값을 갖는다. 또한 audios는 해당 video4linux 디바이스 내의 오디오 디바이스를 뜻한다.

캡쳐가 가능한 장치일 경우 갖게 되는 영역의 최대/최소 크기는 범위를 뜻하는 것이 아니다. 캡쳐 영역은 TV의 입력신호(NTSC, PAL, 또는 SECAM)에 의해 영향을 받을 수 있으며, 디바이스의 지원여부에 따라 범위 내에서 몇 가지 특정 영역만이 캡쳐 가능 영역이 된다.
위의 항목 중 가장 중요한 항목은 type 항목 이다. 위 구조체에서 type 항목은 해당 디바이스의 기능을 나타내주는 플래그(flag)의 집합이다. Type 항목이 갖게 되는 값은 여러가지 기능을 뜻하는 비트의 조합으로 낮은 비트부터 차례로 나열하면 다음과 같다.

비트	상수명	의미
1	VID_TYPE_CAPTURE	메모리로 캡쳐 가능
2	VID_TYPE_TUNER	튜너가 장착됨
4	VID_TYPE_TELETEXT	Teletext 기능을 할 수 있음
8	VID_TYPE_OVERLAY	프레임 버퍼로 오버레이 가능
16	VID_TYPE_CHROMAKEY	크로마키에 의한 오버레이가능
32	VID_TYPE_CLIPPING	오버레이 클립핑(Clipping) 지원
64	VID_TYPE_FRAMERAM	프레임 버퍼 메모리 갱신(Overwrite)가능
128	VID_TYPE_SCALES	이미지의 크기 변환 가능
256	VID_TYPE_MONOCHROME	그레이 스케일(Grey Scale)만 사용함
512	VID_TYPE_SUBCAPTURE	일부분만을 캡쳐 할 수 있음
1024	VID_TYPE_MPEG_DECODER	MPEG스트림(Stream)디코딩할 수 있음
2048	VID_TYPE_MPEG_ENCODER	MPEG 스트림을 인코딩할 수 있음
4096	VID_TYPE_MJPEG_DECODER	MJPEG 스트림을 디코딩할 수 있음
8192	VID_TYPE_MJPEG_ENCODER	MJPEG 스트림을 인코딩할 수 있음

위에 나열된 각 타입은 type 항목을 해당 상수로 마스킹(&연산을 취함)하여 얻은 값에 따라 알아낼 수 있다. 즉, 해당 플래그가 1을 나타내면 해당 기능을 지원하는 것 이다.

* 오버레이란 여러 장의 그림을 겹쳐 놓는 것을 말하며, 비디오 오버레이는 그래픽 카드에서 발생한 데이타와 외부의 장치에서 입력된 영상 데이타를 혼합하여 컴퓨터 모니터에 표시하는 기술을 말한다.

* 크로마키란 두개의 화상을 오버레이 하여 지정된 색에 대해 투과율을 조정해서 화상을 합성시켜 표시한다.

3. Video Source

video4linux의 디바이스들은 적어도 하나 이상의 소스를 갖고 있으며, 이는 채널로 분류된다. 어떤 디바이스라도 사용하기 전에 사용할 채널을 결정하여야 하며, 이는 앞에서 말한 것과 같이 사용 환경에 의해 변할 수 있으므로 채널의 정보를 먼저 알아 보아야 한다. 기본적으로 디바이스에 존재하는 채널의 수는 앞의 VIDIOCGCAP ioctl Call에 의해 알아낼 수 있으며 채널은 0번 부터 시작하게 된다.
각 채널의 정보는 struct video_channel의 channel을 미리 설정한 후 다음의 ioctl Call에 의해 얻을 수 있다.

```
---------------------------------------------------------------------------
 #include <errno.h>
 #include <sys/types.h>
 #include <sys/ioctl.h>
 #include <linux/videodev.h>

 int ioctl(int filedes, VIDIOCGCHAN, struct video_channel *);
---------------------------------------------------------------------------
```

기본적으로 이 호출 또한 성공할 경우 0을, 실패할 경우 errno.h에 정의 된 에러 코드의 음수 값을 반환한다. 따라서 0보다 작은 값이 반환될 경우 오류가 발생한 것이다.

VIDIOCGCHAN ioctl Call을 사용하기 전에는 반드시 구조체 video_channel의 항목 중 channel을 올바른 채널 번호로 설정하여야 한다. 그렇게 해야만 해당 채널에 대한 정보를 얻을 수 있으며, 채널이 올바르지 않다면 오류가 발생될 것이다.

VIDIOCGCHAN ioctl Call이 성공적으로 수행되었다면 struct video_channel에는 해당 채널에 대한 정보가 담겨지게 된다. 이 때 struct video_channel은 다음과 같이 정의 되어 있다.

자료형	이름	의미
int	channel	채널의 번호
char [32]	name	입력의 이름으로 카드 입력의 레이블(Label)이 반영
int	tuners	이 채널에 연결된 튜너의 수
__u32	flags	입력이 갖고 있는 속성
__u16	type	입력의 타입

__u16 norm 이 채널의 표준 (TV의 신호 모드)

위의 항목 중 마지막 세 가지 항목은 video_capability의 type 항목처럼 비트 플래그의 조합으로 구성되어 있다. 따라서 마스킹을 통해 그 의미를 파악할 수 있다.

flags 항목의 의미는 다음과 같다.

비트	상수명	의미
1	VIDEO_VC_TUNER	채널이 튜너를 갖고 있음
2	VIDEO_VC_AUDIO	채널에 오디오가 있음

type 항목의 의미는 다음과 같다.

비트	상수명	의미
1	VIDEO_TYPE_TV	TV 입력
2	VIDEO_TYPE_CAMERA	카메라 입력

norm 항목은 TV 신호 모드를 뜻하며 다음과 같이 정의 되어 있다.

비트	상수명	의미
0	VIDEO_MODE_PAL	PAL 모드
1	VIDEO_MODE_NTSC	NTSC 모드
2	VIDEO_MODE_SECAM	SECAM 모드
3	VIDEO_MODE_AUTO	모드를 설정할 수 없거나, 자동 변환 함

위에서 VIDIOCGCHAN ioctl Call을 통해 얻은 정보를 토대로 VIDIOCSCHAN을 호출함으로서 캡쳐 하고자 하는 채널을 선택할 수 있다. 채널을 선택할 때는 channel과 norm 항목이 올바르게 설정된 video_channel 구조체를 인수로 하여 호출하게 된다. 즉 각각의 채널은 각각의 설정을 갖고 있으며, 해당 설정을 정확히 설정하여 캡쳐하고자 하는 채널을 선택하여야 한다.

```
#include <errno.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/videodev.h>

int ioctl(int filedes, VIDIOCSCHAN, struct video_channel *);
```

---

4. Capture Windows

캡쳐가 가능한 디바이스일 경우 캡쳐하는 영역은 struct video_window를 사용해 결정한 후 다음의 ioctl Call을 사용해 설정할 수 있다. 물론 영역의 크기는 해당 디바이스의 지원 범위내에 있어야 한다.

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/videodev.h>

int ioctl(int filedes, VIDIOCSWIN, struct video_window *);
```

이 구조체는 다음의 구조로 이루어져 있으며, 캡쳐하는 영역과 필요하다면 추가로 클리핑(Clipping) 영역을 설정할 수 있다.

자료형	이름	의미
__u32	x	X 윈도우에서의 X 좌표
__u32	y	X 윈도우에서의 Y 좌표
__u32	width	캡쳐 할 이미지의 가로 폭
__u32	height	캡쳐 할 이미지의 세로 높이
__u32	chromakey	크로마키의 값 (RGB32 값)
__u32	flags	추가적인 캡쳐 플래그
struct video_clip *	clips	클리핑 할 직사각형의 리스트 (Set only)
int	clipcount	클리핑 할 직사각형의 수 (Set only)

flags 항목의 의미는 다음과 같다.

비트	상수명	의미
1	VIDEO_WINDOW_INTERLACE	인터레이스
16	VIDEO_WINDOW_CHROMAKEY	크로마 키

위에서 추가적 정보인 클리핑 할 직사각형의 리스트 clips는 struct video_clip의 배열 형태로 넘겨주게 된다. video_clip의 구조체는 다음과 같이 정의 된다.

자료형	이름	의미
__s32	x	스킵(Skip) 할 직사각형의 X 좌표
__s32	y	스킵 할 직사각형의 Y 좌표
__s32	width	스킵 할 직사각형의 가로 폭
__s32	height	스킵 할 직사각형의 세로 폭
struct video_clip *	next	사용자나 드라이버가 사용만 할 수 있는 요소로 설정하지 않아도 관계 없음

물론 위에서 설정한 내용을 확인하는 호출도 존재하는데 이는 다음과 같다. 현재 캡쳐 윈도우의 설정 값을 가져 올 때는 clips와 clipcount는 무시 된다. 즉, 이 두가지 요소는 설정만 가능하다.

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/videodev.h>

int ioctl(int filedes, VIDIOCGWIN, struct video_window *);
```

위에서 정의한 작업은 단지 캡쳐를 하기 위해 위도우 설정을 한 것일 뿐이며, 설정을 한 후 자동적으로 캡쳐가 이루어 지지는 않는다. 만약 오버레이를 지원하고 이를 이용한 캡쳐를 시작하기 위해서는 다음의 호출을 사용하여 1을 넘겨 주어야 하여야 한다. 또한 시작한 캡쳐를 중지하기 위해서는 0을 넘겨 주면 된다.

```
#include <sys/ioctl.h>
#include <linux/videodev.h>

int ioctl(int filedes, VIDIOCCAPTURE, int *);
```

캡쳐 윈도우를 설정하고 위의 호출을 통해 오버레이를 시작하면, 해당 디바이스의 비디오 스트림(Stream)은 직접 비디오 카드의 프레임 버퍼로 오버레이 될 것이다.

몇몇 캡쳐 디바이스에서는 실제로 보이는 이미지의 일부분을 캡쳐할 수 있는 기능을 지원하기도 한다. 즉 해당 디바이스에서 VID_TYPE_SUBCAPTURE 플래그의 값이 1이 라면 이 디바

이스는 이러한 기능을 지원한다고 말할 수 있다. 서브 캡쳐를 하기 위해서는 다음의 호출을 사용하여 서브 캡쳐를 할 영역과 속성을 설정하면 된다.

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/videodev.h>

int ioctl(int filedes, VIDIOCSCAPTURE, struct video_capture *);
```

위에서 사용되는 struct video_capture의 정의는 다음과 같다.

자료형	이름	의미
__u32	x	그랩(Grab) 할 영역의 X 좌표
__u32	y	그랩 할 영역의 Y 좌표
__u16	width	그랩 할 영역의 가로 폭
__u16	height	그랩 할 영역의 세로 높이
__u16	decimation	1초간에 스킵(Skip) 할 프레임 수
__u16	flags	그랩 할 때의 플래그

위의 flags 항목에 사용될 수 있는 값은 다음과 같으며 역시 비트 플래그의 형태를 띈다.

비트	상수명	의미
0	VIDEO_CAPTURE_ODD	홀수 프레임만 캡쳐
1	VIDEO_CAPTURE_EVEN	짝수 프레임만 캡쳐

비디오 윈도우의 경우와 마찮가지로 비디오 서브 캡쳐를 위해 설정된 값은 다음의 호출을 이용하여 얻어올 수 있다.

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/videodev.h>

int ioctl(int filedes, VIDIOCGCAPTURE, struct video_capture *);
```

5. Image Properties

화상(Picture)의 이미지 속성은 다음의 호출을 사용하여 얻을 수 있다. 얻어 온 값은 struct video_picture 형태의 구조체에 저장 된다.

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/videodev.h>

int ioctl(int filedes, VIDIOCGPICT, struct video_picture *);
```

위에서 사용된 구조체는 다음과 같이 정의 할 수 있다.

자료형	이름	의미
__u16	brightness	화상의 밝기
__u16	hue	화상의 색조(hue) (칼라)
__u16	colour	화상의 색(Color) (칼라)
__u16	contrast	화상의 대조(Contrast)
__u16	whiteness	백색도(Whiteness) (그레이 스케일)
__u16	depth	캡쳐의 깊이(Capture Depth)
__u16	palette	이 이미지에서 사용 될 팔레트

기본적으로 palette 항목을 제외 한 다른 모든 요소는 0~65535 사이의 값으로 설정되어 지며, palette는 다음과 같은 값을 갖을 수 있다.

비트	상수명	의미
1	VIDEO_PALETTE_GREY	0~255로 표현되는 선형적인 그레이 스케일
2	VIDEO_PALETTE_HI240	BT848에서 사용하는 8Bit 칼라 큐브(Cube)
3	VIDEO_PALETTE_RGB565	RGB565를 16비트 워드(Word)에 채움(Packed)
4	VIDEO_PALETTE_RGB555	RGB555를 16비트 워드(Word)에 채움(Packed)
5	VIDEO_PALETTE_RGB24	RGB888을 24비트 워드(Word)에 채움(Packed)
6	VIDEO_PALETTE_RGB32	RGB888을 하위 3바이트에 넣은 32bit
7	VIDEO_PALETTE_YUV422	YUV422의 비디오형태 4:2:2의8비트
8	VIDEO_PALETTE_YUYV	.
9	VIDEO_PALETTE_UYYY	표준으로 적합
10	VIDEO_PALETTE_YUV420	YUV420 캡쳐

11	VIDEO_PALETTE_YUV411	YUV411 캡쳐
12	VIDEO_PALETTE_RAW	RAW 캡쳐 (BT848)
13	VIDEO_PALETTE_YUV422P	YUV 4:2:2 Planar
14	VIDEO_PALETTE_YUV411P	YUV 4:1:1 Planar
15	VIDEO_PALETTE_YUV420P	YUV 4:2:0 Planar
16	VIDEO_PALETTE_YUV410P	YUV 4:1:0 Planar
13	VIDEO_PALETTE_PLANAR	Planar 항목의 시작
7	VIDEO_PALETTE_COMPONENT	Component 항목의 시작

위에서 설명된 화상의 이미지 속성은 다음의 호출로 설정할 수 있다.

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/videodev.h>

int ioctl(int filedes, VIDIOCSPICT, struct video_picture *);
```

6. Reading Images

이미지를 읽어 오기 위해서는 두가지의 방법이 존재한다. 한가지는 전형적인 'read' 호출을 이용하는 것이고, 또 다른 한가지는 MMIO(Memory Mapped Input/Output)를 사용하는 것이다. 그러나 두 가지 방법 모두는 디바이스에 의존적이기 때문에 사용 가능할 수도 그렇지 않을 수도 있다.

첫번째로 'read' 호출을 이용하는 방법은 하나의 프레임을 담기 위한 버퍼를 마련하고, 디바이스로부터 직접 하나의 프레임을 가져온다.

```
#include <sys/types.h>

size_t read(int filedes, void *buf, size_t nbyte);
```

두번째 MMIO를 사용한 방법은 특정 디바이스나 파일을 메모리의 일정 공간으로 매핑(Mapping)하여 메모리 입/출력과 동일 하게 연산할 수 있도록 한 시스템을 말한다. MMIO를 사용하기 위해서는 우선 다음 호출을 통해 video4linux 디바이스의 버퍼와 관련된 정보를 얻어와야 한다.

```
#include <sys/ioctl.h>
#include <linux/videodev.h>

int ioctl(int filedes, VIDIOCGMBUF, struct video_mbuf *);
```

위에서 사용된 구조체 video_mbuf는 다음과 같이 정의할 수 있다.

자료형	이름	의미
int	size	매핑(Mapping) 하려는 메모리의 용량(Bytes)
int	frames	프레임(Frames)의 수
int	offsets[VIDEO_MAX_FRAME]	각 프레임의 오프셋(Offset)

디바이스의 정보를 얻어온 후 이를 이용하여 메모리에 매핑(Mapping)을 하여야 한다. 이를 위해서는 다음의 호출을 이용하여야 한다.

```
#include <sys/types.h>
#include <sys/mman.h>

void *mmap(void *addr, size_t mbuf.size, PROT_READ, MAP_SHARED, int filedes, 0);
```

위의 호출에 관한 자세한 정보는 시스템 콜 'mmap'와 관련된 자료를 참고하기 바란다.

메모리에 성공적으로 매핑 된 후 VIDIOCMCAPTURE 호출을 이용하여 이미지정보를 설정하고 버퍼로 캡쳐를 시작한다. 이를 위해 사용되는 구조체는 다음과 같이 구성되어 있다.

자료형	이름	의미
unsigned int	frame	더블 버퍼링을 위한 캡쳐 할 프레임 번호
int	width	이미지의 폭
int	height	이미지의 높이
unsigned int	format	앞서 설명된 palette 값을 갖음

위의 구조체를 이용한 VIDIOCMCAPTURE 호출은 다음과 같이 사용된다.

```
#include <sys/ioctl.h>
#include <linux/videodev.h>

int ioctl(int filedes, VIDIOCMCAPTURE, struct video_mmap *);
```

또한 캡쳐를 마무리 하고 사용하던 프레임의 사용권을 반환하기 위해서 다음의 호출을 이용한다.

```
#include <sys/ioctl.h>
#include <linux/videodev.h>

int ioctl(int filedes, VIDIOCSYNC, int *);
```

또한, 모든 메모리 맵의 사용이 완료되면 매핑(Mapping)을 해제하기 위해 다음의 호출을 이용한다.

```
#include <sys/types.h>
#include <sys/mman.h>

int munmap(void *addr, size_t len);
```

7. References

[1] Alan Cox, Video4Linux API Kernel Documents, http://pl.smu.ac.kr/researches/projects/linux-dvr/v4l-kernel-doc

[2] Maxwell Sayles, "How To" for Video For Linux (VFL), http://pages.cpsc.ucalgary.ca/~sayles/VFL_HowTo/, 2002.

[3] 서영진, Linux Multimedia Programming, http://user.chollian.net/~valentis/Suhdang/QT_Programming/Lecture_MM.html, 2003.

제2절 video4linux를 사용하여 비디오 캡쳐하기

본 절에서는 v4l을 사용하여 비디오 캡쳐(video capture)하는 방법을 각 단계별로 예제 소스를 가지고 간략히 설명한다.

1. video4linux란 무엇인가?

video4linux는 리눅스에서 비디오 디바이스를 제어하고 사용하기 위한 API이다. 비디오 디바이스는 튜너를 포함한 비디오 카드나 웹캠과 같이 영상 정보를 입력할 수 있는 장치를 뜻하며, 튜너를 포함한 디바이스의 경우 텔레비전이나 AM/FM 라디오, 텔레텍스트(Teletext)등의 방송을 시청/청취할 수 있다.

video4linux는 리눅스 커널에 기본적으로 포함되어 있으며, 커널 컴파일을 통해 커널 모듈, 혹은 정적으로 컴파일하여 사용이 가능하다.

video4linux는 각 디바이스를 위해 다음의 파일을 제공하고 있으며, video4linux를 이용하기 위해서는 다음의 디바이스 파일을 열고(Open)하고 검색하여 지원하는 기능을 차아낸 후 API를 이용하여 사용할 수 있다.

디바이스 이름	마이너 범위	기능
/dev/video	0-63	비디오 캡쳐 인터페이스
/dev/radio	64-127	AM/FM 라디오 디바이스
/dev/vtx	192-223	Teletext 인터페이스 칩
/dev/vbi	224-239	Raw VBI 데이타 (Intercast/teletext)

현재는 video4linux보다 나은 기능과 다양한 디바이스의 지원을 위한 Video For Linux Two가 등장하였다.

- 부연설명
* 텔레텍스트(Teletext)는 문자방송을 뜻한다.
* VBI는 Vertical Blanking Interval의 약자로 화면에 보이지 않지만, 이 부분을 이용하여 Digital Data가 방송 되는 것을 뜻한다.
* 인터캐스트(Intercast)는 VBI를 이용하여 TV와 Internet을 융합한 시스템을 말한다.

2. 비디오 캡쳐 들어가기(Video Capture Overview)

이번 장에서는 우리가 비디오 캡쳐(video capture)를 하기 위해서 수행해야 할 일을 단계별로

간략히 살펴본다. 각 단계는 다음과 같다.

---

1 단계: 작업 환경
2 단계: 비디오 디바이스와 통신하기
3 단계: 캡쳐를 할 수 있는가?
4 단계: 어떤 입력(channel)을 선택할 것인가?
5 단계: 어떤 스케치북을 사용하나?
6 단계: 물감을 어떻게 사용할까?
7 단계: 이미지 읽어 오기
8 단계: 이미지 포맷 변환
9 단계: PPM으로 저장하기
10 단계: 정리하기

비디오 캡쳐를 하기 위한 과정은 위와 같이 총 열 단계로 이루어져 있다. 1 단계는 프로그램을 하기 전에 작업 환경을 살펴보고, 2 단계 부터 10 단계까지는 실질적인 프로그램을 할 수 있도록 도와준다. 다음 장 부터 각 단계별 설명을 한다.

본 문서에서 제공하는 최종 소스 코드는 video-capture.c, video-capture.h, 그리고 Makefile 이다. 또한, 각각의 경우에 따라 4가지의 디렉토리로 분리되어 있다.

- 'read' 시스템 콜로 읽어오고 이미지 포맷은 YUV420P를 사용할 경우
- 'read' 시스템 콜로 읽어오고 이미지 포맷은 RGB24를 사용할 경우
- 'mmap' 시스템 콜로 읽어오고 이미지 포맷은 YUV420P를 사용할 경우
- 'mmap' 시스템 콜로 읽어오고 이미지 포맷은 RGB24를 사용할 경우

최종 소스 코드는 디렉토리로 나뉘어진 부분이 옵션으로 이루어져 있으며, 사용법은 명령어의 usage를 참조하라.

3. 작업 환경

이번장에서는 우리가 문서를 작성하면서 첨부한 소스코드의 실행 환경에 대해서 소개하겠다.

3.1 컴퓨터 정보

CPU: Pentium II 266Hz
Memory: 128M
OS: Redhat linux 9.0

kernel: Linux version 2.4.20-8
Gcc version: 3.2.2
HDD: 4GB

3.2 Webcam

Type: USB
Name: AlphaCam i
DeviceDriverName: OV511+ USB Camera

3.3 디바이스 확인하기

우리가 사용한 디바이스는 USB디바이스로 컴퓨터와 연결시킬 경우 모듈이 바로 올라간다. 하지만 가끔 올라가지 않는 디바이스가 있을 경우도 있다. 이럴 경우에 우리는 모듈이 올라갔는지를 확인해야 한다.
모든 비디오 디바이스는 디바이스 드라이버를 올릴 경우 videodev가 같이 올라가게 된다. 따라서 우리는 videodev모듈이 올라 왔는지를 확인하면 된다.

```
[root@elf /]# lsmod
Module                  Size  Used by    Tainted: PF
ov511                  82304   0
pwcx-2.4.20            88032   0  (unused)
pwc                    47688   0  [pwcx-2.4.20]
videodev                8288   1  [ov511 pwc]
ide-cd                 35708   0  (autoclean)
cdrom                  33728   0  (autoclean) [ide-cd]
parport_pc             19076   1  (autoclean)
lp                      8996   0  (autoclean)
parport                37056   1  (autoclean) [parport_pc lp]
autofs                 13268   0  (autoclean) (unused)
eepro                  16424   2
ipt_REJECT              3928   2  (autoclean)
iptable_filter          2412   1  (autoclean)
ip_tables              15096   2  [ipt_REJECT iptable_filter]
keybdev                 2944   0  (unused)
mousedev                5492   1
hid                    22148   0  (unused)
input                   5856   0  [keybdev mousedev hid]
```

```
usb-uhci                26348   0  (unused)
usbcore                 78784   1  [ov511 pwc hid usb-uhci]
ext3                    70784   2
jbd                     51892   2  [ext3]
[root@elf /]#
```

3.4 모듈올리기

만약 3.3처럼 했는데 디바이스의 목록에 videodev라는 항목이 없으면 디바이스가 올라가지 않은 것이다.(videodev는 비디오 디바이스를 올릴 경우 같이 올라가는 가상 드라이버이다.)
이 같은 경우 우리는 디바이스의 드라이버를 사용하여 모듈을 올려줘야 한다. 올리는 방법은 다음과 같다.

```
[root@elf /]# modprobe pwc
[root@elf /]#
```

위 내용의 pwc는 Philips Webcam driver의 준말로서 디바이스 드라이버 이름이다.
여기서 사용된 pwc 디바이스 드라이버는 제한된 기능들이 많이 있으므로, 우리는 pwcx를 추가적으로 다운받아 사용하기로 한다.
디바이스 드라이버 파일은 http://www.smcc.demon.nl/webcam/release.html 여기서 최근 pwcx 디바이스 드라이버 파일을 다운로드 한다. 그런 후 tar-zxvf [파일이름]을 통해 압축을 푼 후 자신의 커널 버전에 맞는 디렉토리의 *.o 파일을 '/lib/modules/[자기커널버젼]/kernel/drivers/usb/'아래에 넣도록 한다. 그런 후 위에서 설명한 것처럼 모듈을 올려주면 된다.
지원되는 디바이스 드라이버의 목록은 "/lib/modules/[자기커널버전]/kernel/drivers"아래에 있다.

```
[root@canna usb]# pwd
/lib/modules/2.4.20-8/kernel/drivers/usb
[root@canna usb]# ls
CDCEther.o    brlvger.o   hid.o          microtek.o    pwc.o
se401.o       uhci.o      usbcore.o      vicam.o
acm.o         catc.o      hpusbscsi.o    ov511.o       pwcx-2.4.20.o
serial        ultracam.o  usblcd.o       wacom.o
aiptek.o      dabusb.o    ibmcam.o       pegasus.o     rio500.o
```

```
storage      usb-midi.o   usbnet.o
audio.o        dsbr100.o   kaweth.o       powermate.o   rtl8150.o
stv680.o    usb-ohci.o   usbvideo.o
auerswald.o   hcd          mdc800.o       printer.o      scanner.o
tiglusb.o   usb-uhci.o   uss720.o
[root@canna usb]#
```

3.5 필요한 헤더파일

v4l 디바이스를 통신하기 위해서는 많은 함수와 구조체와 상수등이 필요하다. 따라서 이러한 것들을 사용하기 위해서 다음과 같은 헤더파일들을 소스코드의 가장 윗부분에 추가한다.

추가해야하는 헤더파일

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <linux/videodev.h>
#include <math.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include "video-capture.h"
```

---

4. 비디오 디바이스와 통신하기

4.1 비디오 디바이스 열기

우리가 이제부터 해보고자 하는 프로그램은 비디오 디바이스로부터 정보를 받아 컴퓨터에 저장하는 것이다. 이를 위해 가장 먼저 해야 할 일은 우리가 만들 프로그램이 비디오 디바이스와 정보를 주고 받을 수 있게 연결해 주는 것이다.

디바이스를 사용하기 위해 우선 디바이스 파일을 열어 디바이스에 대한 접근 경로를 구해야한다. 이를 위해 open 시스템 콜을 사용한다.

```
#include<sys/types.h>                  /* mode_t 정의 */
#include<fcntl.h>                      /* open, close 정의 */
```

```
int video;
if ( (video = open(device, O_RDONLY)) < 0 )
{
  fprintf(stderr, "%s: video device %s could not be openedWn",
  cmd, device);
  exit(1);
}
```

위 코드는 명령행 인수로 디바이스 파일을 입력받아서 해당 디바이스를 open하고 파일 기술자를 받아 video변수에 저장한다. 파일 기술자는 이후 디바이스와 통신을 위해 계속 사용하게 된다.

4.2 디바이스 입출력 제어

디바이스에 대한 접근 경로를 구했으니 이제 디바이스를 제어할 준비가 됐다. 디바이스의 동작을 제어하고 속성을 설정하고, 구하기 위한 ioctl 인터페이스가 있다.
ioctl에 인수는 제어할 디바이스의 디바이스 파일의 기술자와 디바이스에 보낼 명령이 필요하다. 파일 기술자는 이전에 디바이스 파일을 open하고 반환받은 값을 준다.
그리고 명령은 부록 A v4l API에 정의된 것들을 사용하면 된다. 이에 대한것은 다음 장부터 설명할 것이다. 더 자세한 사항은 부록 A v4l API를 참조한다. 세번째 인수는 ioctl 콜로 디바이스에 보낸 명령에 의해 받은 속성을 저장하거나 설정하기 위해 속성을 저장한 구조체를 쓴다.

```
#include<sys/ioctl.h>              /* ioctl 정의 */
```

```
ioctl(video, VIDIOCGCAP, &cap)
```

위 코드는 비디오 디바이스로부터 디바이스에 대한 정보를 받아온다. Ioctl 콜로 디바이스에 VIDIOCGCAP 명령과 디바이스에 대한 정보를 저장할 구조체 video_capability의 주소를 전달하면 디바이스의 정보를 읽어 구조체에 저장한다.

더 자세한 설명은 다음 장에서 나올 것이므로 여기서는 ioctl을 사용하는 형식만 알면 된다.

5. 캡쳐를 할 수 있는가?

비디오 다바이스(Video Device)를 열었다면, 관련된 정보를 얻을수 있다. 이장에서는 그런 정보를 얻는 방법을 기술한다.

5.1 디바이스 정보 읽어 오기

상용 프로그램(application)이 아니라면, 굳이 디바이스(device)정보를 보여주지 않을 수도 있다. 하지만 여러 디바이스(device)가 동시에 동작하는 프로그램(program)이 수행되는 환경이라면, 특정 디바이스(device)를 지정하기 위해서 디바이스(device) 정보를 읽어온다.
캡쳐(capture)를 제공하는지 알아보거나, 캡쳐(capture)가능한 픽셀(pixels) 수의 범위등이 디바이스(device)정보의 범주에 들어갈 것이다.

다음의 관련 소스는 v4l 디바이스(device)로부터 정보를 얻어 오는 것이다.

```
---------------------------------------------------------------------------
return_value = ioctl(video , VIDIOCGCAP, &cap);
if ( return_value < 0 )
{
  fprintf(stderr, "%s: ioctl on VIDIOCGCAP failed₩n", cmd);
  exit(1);
}
---------------------------------------------------------------------------
```

struct video_capability에 관한 내용은 부록 A v4l API를 참고하라.
VIDIOCGCAP ioctl은 video_capability에 디바이스(device)관련 정보를 알려준다. ioctl호출 실패시에 관련된 음수값을 반환한다.

얻은 장치(device) 정보는 다음과 같이 확인할 수 있다.

```
---------------------------------------------------------------------------
printf("The video capabilities are as follows:₩n");
printf("name of the interface is %s₩n", cap.name);
/*
 * Check the type
 * Reference <linux/videodev.h>
 */
if ( !(cap.type & VID_TYPE_CAPTURE) )
{
  fprintf(stderr, "%s: can not capture(VID_TYPE_CAPTURE is false)₩n", cmd);
  exit(1);
```

```
}
else
{
  printf("can capture(VID_TYPE_CAPTURE)₩n");
}

if ( cap.type & VID_TYPE_SCALES )
{
  printf("scalable(VID_TYPE_SCALES)₩n");
}

if ( cap.type & VID_TYPE_SUBCAPTURE )
{
  printf("can capture subareas of the image(VID_TYPE_SUBCAPTURE)₩n");
}

printf("number of channels present is %d₩n", cap.channels);
printf("number of audios present is %d₩n", cap.audios);
printf("max capture width is %d pixels₩n", cap.maxwidth);
printf("max capture height is %d pixels₩n", cap.maxheight);
printf("min capture width is %d pixels₩n", cap.minwidth);
printf("min capture height is %d pixels₩n", cap.minheight);
```

결과값

```
The video capabilities are as follows
name of the interface is OV511 + USB Camera
can capture(VID_TYPE_CAPTURE)
can capture subareas of the image(VID_TYPE_SUBCAPTURE)
number of channels present is 1
number of audios present is 0
max capture width is 640 pixels
max capture height is 480 pixels
min capture width is 64pixels
min capture height is 48 pixels
```

---

6. 어떤 입력(channel)을 사용할 것인가?

6.1 디바이스와 채널의 관계

v4l 디바이스(device)의 종류중에 Webcam과 TV 인터페이스(Interface)가 있다. 이중 전자인 Webcam의 경우에는 입력신호를 본체의 렌즈 1개에서만 받아들인다. 때문에 입력 신호를 처리하는 채널(channel)도 1개만 존재한다. 반면에 TV 인터페이스(Interface)는 여러 입력(input)에서 신호를 다룬다. 때문에 입력 신호를 처리하는 채널(channel)이 여러 개 존재할 수 있다. 이 장에서는 먼저 이러한 채널(channel)의 정보를 조사한다. 조사한 후에 자신의 프로그램(program)에 맞추어 특정값을 입력하는 내용을 다룬다. 마지막으로는 구조체 vidieo_channel의 필드 'norm'에 해당되는 비디오 레코딩(Video Recording)의 표준 방식에 대해서 설명한다.

6.2 장치에서 사용가능한 비디오 채널 정보 보여주기

채널(channel) 열거후에 user에게 특정 채널(channel)을 선택하고 싶다거나, 이미 채널(channel) 번호를 알고 있다면, 나중에 비디오 디바이스(video device)가 직접 사용하는 채널(channel)을 선택할 수 있다.

다음의 관련 소스는 v4l 디바이스(device)로부터 1번 채널(channel)의 정보를 얻어 온다.

```
-------------------------------------------------------------------------
chan.channel = i;               /* i = 1 */
return_value = ioctl(video, VIDIOCGCHAN, &chan);
if ( return_value < 0 )
{
  fprintf(stderr, "%s: ioctl on VIDIOCGCHAN failedWn", cmd);
  exit(1);
}
-------------------------------------------------------------------------
```

struct video_channel에 관한 내용은 부록 A v4l API를 참고한다.
VIDIOCGCHAN ioctl은 video_channel에 디바이스(device)관련 정보를 알려준다. ioctl호출 실패시에 관련된 음수값을 반환한다.

얻은 device 정보는 다음과 같이 확인할 수 있다.

```
-------------------------------------------------------------------------
for ( i = 0; i < cap.channels; i++ )
{
  chan.channel = i;
  return_value = ioctl(video, VIDIOCGCHAN, &chan);
```

```
if ( return_value < 0 )
{
  fprintf(stderr, "%s: ioctl on VIDIOCGCHAN failed₩n", cmd);
  exit(1);
}

printf("name of the channel is %s₩n", chan.name);
printf("number of tuners for this input is %d₩n", chan.tuners);

/* Test channel.flags */
if ( 0 == chan.flags )
{
  printf("properties channel.falgs are not defined₩n");
}
else
{
  if ( chan.flags & VIDEO_VC_TUNER )
  {
    printf("Channel has a tuner(VIDEO_VC_TUNER)₩n");
  }

  if ( chan.flags & VIDEO_VC_AUDIO )
  {
    printf("Channel has audio(VIDEO_VC_AUDIO)₩n");
  }
}

/* Test channel.type */
if ( chan.type & VIDEO_TYPE_TV )
{
  printf("TV Input(VIDEO_TYPE_TV)₩n");
}

if ( chan.type & VIDEO_TYPE_CAMERA )
{
  printf("Camera Input(VIDEO_TYPE_CAMERA)₩n");
}

/* Test channel.norm */
```

```
  if ( VIDEO_MODE_PAL == chan.norm ) /* VIDEO_MODE_PAL is 0 */
  {
    printf("PAL(VIDEO_MODE_PAL)₩n");
  }
  else if ( VIDEO_MODE_NTSC == chan.norm ) /* VIDEO_MODE_NTSC is 1 */
  {
    printf("NTSC(VIDEO_MODE_NTSC)₩n");
  }
}
```

결과값

```
name of the channel is Camera
number of tuners for this input is 0
properties channel.flags ar not defined
Camera input(VIDEO_TYPE_CAMERA)
```

6.3 디바이스에 비디오 채널 정보 등록하기

다음은 위에서 읽은 channel의 default정보를 기본으로 해서, 디바이스 타입(device type)과 비디오 레코딩(Video Recording)를 지정하는 소스이다.

```
/*
 * Settings for input channel 1 which is Composite0
 * flags: set no tuenrs and no audio
 */
chan.flags = 0;
chan.type = VIDEO_TYPE_CAMERA;
chan.norm = VIDEO_MODE_NTSC;          /* 부록 B Terms 참조 */
if( (ret = ioctl(deviceHandle, VIDIOCSCHAN, &chan)) < 0 )
{
    fprintf(stderr, "ioctl on VIDIOCSCHAN failed₩n");
    exit(1);
}
```

VIDIOCGCHAN ioctl은 비디오 채널(video_channel)에 디바이스(device)관련 정보를 알려준다.

7. 어떤 스케치북을 사용하나?

지금까지 우리는 입력신호를 선택하고, 채널을 설정하는 것을 배웠다. 이제부터 우리가 할 것은 설정된 신호가 들어와 메모리에 쓰여 질 때, 쓰여지는 종이의 크기를 결정하는 것이다. 이러한 정보들은 video_window라는 구조체에 정의 되어 있다. video_window구조체에는 많은 정보 값들을 가지고 있지만 우리가 가장 많이 사용하는 부분은 역시 종이의 가로와 세로의 길이를 나타내는 width와 height가 될 것이다 자 이제 구조체에서 우리가 필요한 종이의 가로, 세로 정보를 읽어오는 방법을 알아보자.

7.1 변수와 구조체 선언하기

우리는 모든 변수를 사용할 때 선언을 해야 한다. 구조체 또한 변수들의 집합이므로 이러한 구조체를 사용하기에 앞서 먼저 구조체를 선언해야 한다. 왜냐하면 디바이스에서 가져오는 가로, 세로의 정보를 넣어줄 메모리의 영역이 필요하기 때문이다 또한 구조체에서 가져온 정보를 담을 지역변수 또한 선언해 주어야 한다.
변수와 구조체의 선언은 다음과 같다.

```
---------------------------------------------------------------------------
int width, height;
struct video_window win;
---------------------------------------------------------------------------
```

이러한 구조체의 선언은 항상 소스코드의 앞부분에서 이루워 져야 하므로 소스코드의 가장 앞부분에 선언 되어야 한다. 우리는 'win' 구조체에서 이미지의 가로와 세로의 길이만 얻어올 것이므로 width와 height만 지역변수로 선언한다. 만약 다른 구조체의 값들이 필요하다면 지역변수 또한 추가로 설정해 주어야 한다.

7.2 video_window 구조체에 디바이스의 정보 가져오기

구조체의 선언이 되어 있다면, 이제 우리가 해야 할일은 선언되어 있는
'win' 구조체에 값을 채워 주는 일이다 이러한 일은 ioctl함수를 사용해서 할 수 있다.

```
---------------------------------------------------------------------------
return_value = ioctl( video, VIDIOCGWIN, &win );
if ( return_value < 0 )
{
  fprintf(stderr,"%s: could not obtain specifics of capture
  window₩n", cmd);
  exit(1);
```

}

---

ioctl함수를 사용해 'win' 구조체에 값을 할당 하는데 문제가 생길 경우를 대비해 우리는 ioctl 함수의 반환값을 확인한다. 만약 반환값이 음수이면, 우리는 ioctl함수를 사용하여 디바이스에서 'win' 구조체로의 정보를 채우는 일이 실패했음을 알려 줘야 한다.

```
#ifdef DEBUG
printf("Properties for image capture area:₩n");
printf("x = %d", win.x);
printf(", y= %d", win.y);
printf(", width = %d", win.width);
printf(", height = %d", win.height);
printf(", chromakey = %hx", win.chromakey);
printf(", flags = %d₩n", win.flags);
#endif
```

우리는 프로그래밍을 하면서 디바이스가 어떠한 것들을 제공 할 수 있는지 확인하거나 대입한 지역변수가 정확히 들어갔는지 확인 할 필요가 있다. 하지만, 이러한 정보들이 항상 필요한 것은 아니다. 따라서, 여기서는 소스의 처음에 'DEBUG'를 정의함으로써 정보 값들의 확인이 필요할 때만 표준에러로 출력 할 수 있도록 프로그래밍 한다.

우리는 ioctl함수의 2번째 인수에 따라 ioctl함수가 정보를 가져오거나 정보를 설정하도록 만들 수 있다. 여기서는 정보를 가져오는 것만을 다루었다. 정보를 가져오는 것은 VIDIOCGWIN을 2번째 인수에 넣어주면 된다. 만약, 당신이 정보를 설정하고 싶다면 먼저 지역변수에 설정하고 싶은 값을 채워 넣은 다음 ioctl함수의 2번째 인수에 VIDIOCSWIN을 넣은 후 ioctl함수를 불러주면 된다. VIDIOCGWIN과 VIDIOCSWIN같은 것들은 상수인데, 이러한 상수들에 대한 자세한 설명은 부록 A v4l API를 참조하기 바란다

7.3 video_window 구조체에 디바이스의 정보 설정오기

우리는 여기서 width와 height만을 설정할 것이므로 5장에서 사용한 'cap' 구조체를 사용할 것이다. 5장에서 사용한 'cap'구조체의 maxwidth와 maxheight정보 'win' 구조체에 설정하기 위해서 우리는 win.width와 win.height에 cap.maxwith와 cap.maxheight 를 넣어 준다. 그런 후 변경하지 않을 다른 값들은 0으로 설정해 준다. 그런후 ioctl을 사용해서 디바이스의 값들을 'win' 구조체의 내용으로 변경해 주면 된다. 이걸로서 디바이스의 설정값들을 우리는 손쉽게 설정한 것이다

```
width = win.width = cap.maxwidth;
height = win.height = cap.maxheight;
win.x = win.y = win.chromakey = win.flags = 0;
return_value = ioctl( video, VIDIOCSWIN, &win );
if ( return_value < 0 )
{
  fprintf(stderr,"%s: ioctl on VIDIOCSWIN failed₩n", cmd);
  exit(1);
}
```

8. 물감을 어떻게 사용할까?

앞장에서 우리는 디바이스에서 필요한 정보인 종이의 가로와 세로 길이를 가져오는 것에 대해 알아보았다. 이번 장에서는 이렇게 가져온 종이에 필요에 의해서 사용될 물감의 종류에 대해서 설정 하거나 어떠한 물감이 설정되어 있는지를 확인 할 것이다.
이러한 물감의 정보에 대한 것들은 video_picture라는 구조체에 정의 되어 있다. video_picture 구조체 또한, video_window구조체와 마찬가지로 많은 정보를 가지고 있지만 우리는 가장 많이 사용되는 깊이와 팔레트에 대해서 알아볼 것이다.

8.1 용어

8.1.1 깊이

하나의 픽셀에 얼마만큼의 비트가 할당되느냐를 나타낸다. 보통 우리는 24bit를 많이 사용하고 32bit는 24bit의 색상에 알파채널이 추가된 것이다. 자세한 내용은 부록B Terms를 참고 하기 바란다.

8.1.2 팔레트

팔레트중 우리가 주로 많이 사용하는 이미지 포맷으로 RGB와 YUV를 들 수 있다. 더 자세한 팔레트 상수의 내용은 부록A v4l API에서 다루기로 하고 여기서는 RGB와 YUV의 특징에 대해서만 알아보자.

- RGB

칼라가 Red, Green, Blue의 3색의 강도(intensity)를 나타내는 세 쌍 숫자로 표현[(R + G + B) = 1(흰색)]된다. 따라서 칼라 CRT모니터의 R,G,B 전자총의 전압으로 쉽게 맵핑 될 수 있기

때문에 비디오 디스플레이 드라이버에 편리한 모델이다.

- YUV

YUV는 NTSC, PAL, SECAM에 사용되는 텔레비젼 산업의 기본적인 칼라 포맷이다.
현재는 YIQ는 NTSC, YUV는 PAL, SECAM에서 사용되며, YCbCr은 MPEG에서 사용하는 칼라 모델이다. Y는 휘도를 나타내며, IQ, UV, CbCr은 비디오 신호의 색상 부분을 형성한다. 따라서 Y신호가 그레이 레벨을 제공하므로 흑백 텔레비젼과의 호환이 가능하다.
보통 RGB의 영상을 YUV로 압축하는데 이유는 사람의 눈이 휘도에는 민감하나 색상차이에는 휘도보다 민감하지 않기 때문이다. 따라서, Y:U:V를 4:2:2정도로 압축하는데 자세한 내용은 부록B Terms를 참조하기 바란다.

8.2 변수와 구조체 선언하기

우리는 앞장에서 video_window구조체의 값들을 사용하기 위해 지역변수와 구조체를 선언 했다. 마찬가지로 video_picture구조체 또한, 필요한 정보를 얻기위해 선언 해주어야 한다. 또한, 깊이와 팔레트값을 가져오기 위해서 우리는 지역변수 또한 선언한다.
변수와 구조체의 선언은 다음과 같다.

```
---------------------------------------------------------------------
int depth, palette;
struct video_picture pic;
---------------------------------------------------------------------
```

이러한 구조체의 선언은 항상 소스코드의 앞부분에서 이루워 져야 하므로 소스코드의 가장 앞부분에 선언 되어야 한다. 우리는 'pic' 구조체에서 깊이와 팔레트값만을 얻어올 것이므로 깊이와 팔레트만 지역변수로 선언 한다.

8.3 'pic' 구조체에 디바이스의 정보 가져오기

구조체에 정보를 수정했고, 이제 우리는 다시 디바이스에 설정값을 'pic' 구조체에 ioctl함수를 사용하여 가져 옵니다.

```
---------------------------------------------------------------------
return_value = ioctl(video, VIDIOCGPICT, &pic);
if ( return_value < 0 )
{
   fprintf(stderr,"%s: Failed to get the image propertiesWn", cmd);
   exit(1);
```

```
}
```

---

ioctl함수를 사용해 'pic' 구조체에 값을 할당 하는데 문제가 생길 경우를 대비해 우리는 ioctl 함수의 반환값을 확인한다. 만약 반환값이 음수 이면, ioctl함수를 사용하여 디바이스에서 'pic' 구조체로의 정보를 채우는 일이 실패 했음을 알려줘야 한다.

8.4 디바이스의 정보 보기

우리는 프로그래밍을 하면서 디바이스가 어떤 것들을 제공할 수 있는지 확인 할 필요가 있다. 하지만, 이러한 정보들이 항상 필요한 것은 아니다. 따라서, 여기서는 소스의 처음에 'DEBUG' 를 정의함으로써 정보 값들의 확인이 필요할 때만 표준에러로 출력할 수 있도록 프로그래밍 한다.

```
#ifdef DEBUG
   fprintf(stderr, "Image properties are as follows:₩n");
   fprintf(stderr, "brightness = %hx", pic.brightness);
   fprintf(stderr, ", hue = %hx", pic.hue);
   fprintf(stderr, ", color = %hx", pic.colour);
   fprintf(stderr, ", contrast = %hx", pic.contrast);
   fprintf(stderr, ", whiteness = %hx", pic.whiteness);
   fprintf(stderr, ", depth = %hx", pic.depth);
   fprintf(stderr, ", palette = %hx₩n₩n", pic.palette);
#endif
```

위의 내용은 video_picture 구조체의 모든 변수를 표준에러로 출력한 것이다 구조체 내의 변수에 대한 설명은 부록 A v4l API의 video_picture 구조체 부분을 참고 하기 바란다.

8.5 'pic' 구조체의 정보를 변수값에 대입하기

'pic' 구조체에 정보를 채워주는 일이 성공적으로 끝났다면 앞으로 많이 사용될 깊이와 팔레트 값을 지역변수에 대입해 줘야 한다. 이미지 포맷에 따라 pic.palette 값이 다르다. 관련 소스는 아래와 같다.

```
depth = pic.depth = 24; /* image depth is 24bits */
```

RGB24: pic.palette = VIDEO_PALETTE_RGB24;

YUV420P: pic.palette = VIDEO_PALETTE_YUV420P;

8.6 필요하다면 디바이스를 선언한 구조체에 설정된 값으로 수정하기

지금 이야기 하는것은 말 그대로 필요하다면 이라는 전제를 둡니다. 필요할 경우 이미지의 물감 정보를 수정해 준다.
구조체의 선언이 되어 있다면, 이제 우리가 해야 할일은 선언되어 있는 'pic' 구조체에 값을 채워주는 일이다 이러한 일은 미리 채워진 로컬 변수의 값 또는 설정하고 싶은 값을 구조체 변수에 대입 하므로서 가능하다. 다음으로 이렇게 지역변수 또는 설정하고 싶은 값으로 대입되어 있는 'pic' 구조체를 우리는 디바이스의 메모리에 쓰기위해 다시 ioctl함수를 사용한다.

```
return_value = ioctl(video, VIDIOCSPICT, &pic);
if ( return_value < 0 )
{
  fprintf(stderr,"%s: Failed to set the image propertiesWn", cmd);
  exit(1);
}
```

우리의 소스 코드에서는 일단 이미지 물감의 정보를 'pic' 구조체에 가져온다. 다음으로 구조체에 가져오는 것이 성공 하면 메시지를 출력하고, 지역변수 또는 필요한 값으로 구조체의 변수를 설정한다. 마지막으로 우리는 ioctl함수를 사용하여 디바이스에 있는 이미지 물감 정보를 'pic' 구조체에 있는 정보로 수정한다.

여기서 사용된 ioctl의 두번째 인수인 VIDIOCGPICT와 VIDIOCSPICT는 물감정보를 가져오고, 설정하는 상수 값이다 자세한 정보는 부록A v4l API를 참고 하기 바란다

9. 이미지 읽어 오기(read/mmap)

이번 장에서는 이미지를 읽어 오는 방법을 다룬다.

이미지를 읽어 오기 위해서는 두 가지 방법이 있다. 첫 번째 방법은 시스템 콜(system call) 중 하나인 "read"를 사용하는 것이고, 두 번째 방법은 시스템 콜 중 하나인 "mmap"를 사용하는 것이다.

9.1 "read" 시스템 콜 사용

"read" 시스템 콜을 이용하여 이미지를 읽어 오기 위해서는 우선 이미지를 읽어서 저장할 버퍼(buffer)를 할당하고, 다음에 "read" 시스템 콜을 이용하여 프레임(frame)을 읽어 오고, 마지막으로 할당했던 자원을 해제시킨다. 단, 모든 디바이스가 "read" 연산(operations)을 지원하는 것은 아니다.

9.1.1 이미지를 위한 버퍼 할당

이미지를 위한 버퍼 할당시 함수 "malloc"을 사용하고, 이미지 크기는 이미지 포맷을 RGB24로 했을 경우와 YUV420P로 했을 경우가 다르다. 이미지 포맷을 RGB24로 했을 경우 이미지의 폭(width)과 이미지의 높이(height)와 픽셀의 할당된 비트수/8(depth/8)을 곱한 것이다. 픽셀의 할당된 비트수를 8로 나누는 이유는 픽셀당 바이트 값을 얻기 위해서 이다. 이미지 포맷을 YUV420P로 했을 경우 이미지의 폭(width)과 이미지의 높이/4.0(height/4.0)과 6.0을 곱한 것이다. 관련 소스 코드는 다음과 같다.

```
-------------------------------------------------------------------------
RGB24일 경우: int image_size = width * height * (depth/8);
-------------------------------------------------------------------------
YUV420P일 경우: int image_size = (int) (width * height / 4.0 * 6.0);
-------------------------------------------------------------------------
image_buffer = (char*) malloc(image_size);
if ( 0 == image_buffer )
{
  fprintf(stderr, "%s: Image buffer malloc failed.₩n", cmd);
  exit(1);
}
-------------------------------------------------------------------------
```

9.1.2 "read" 시스템 콜를 사용하여 프레임 읽어 오기

프레임을 읽어 오기 위해서 위에서 말한 바와 같이 "read" 시스템 콜를 사용한다. "read" 시스템 콜은 파일 기술자를 이용하여 파일 또는 다른 객체로부터 데이타(data)를 읽기 위해 사용된다.

read 시스템 콜과 위에서 할당한 버퍼(image_buffer)와 이미지 크기(image_size)를 가지고 파일로 부터 이미지를 읽어 온다. 관련 소스는 아래와 같다.

```
-------------------------------------------------------------------------
return_value = read(video, image_buffer, image_size);
```

```
if ( return_value != image_size )
{
  fprintf(stderr, "%s: The return value was not equal to the number
  of requested bytes.₩n", cmd);
  exit(1);
}
```

9.1.3 버퍼 자원 해제

마지막으로 우리는 할당된 버퍼(image_buffer)를 해제시켜야 한다. 이때, 함수 "free"를 사용한다. 관련 소스는 아래와 같다.

```
free (image_buffer);
```

9.2 "mmap" 시스템 콜 사용

"mmap" 시스템 콜을 이용하여 이미지를 읽어 오기 위해서는 우선 "mmap" 인터페이스(interface)를 설정하고, "mmap"을 사용하여 캡쳐하고, 마지막으로 매핑했던 것을 제거한다.

9.2.1 "mmap" 인터페이스 설정

프로세스 메모리에 하드웨어 비디오 버퍼(hardware video buffers를 매핑시키기 위해서는 "mmap" 인터페이스를 사용한다. "mmap" 시스템 콜은 파일 읽기/쓰기를 메모리 읽기/쓰기와 같이 할 수 없을까라는 생각에서 나온 것이라고 한다. 다시 말해서, "mmap" 시스템 콜은 파일 기술자 같은 것으로 표현되는 객체를 가상 주소 상에 매핑 시켜주는 시스템 콜이다. 이것은 파일을 열어서 "mmap"하면 읽기/쓰기를 포인터로 할 수 있게 된다는 의미이다.

"mmap" 인터페이스 설정을 살펴 보면, 세 단계로 나눌 수 있다.

첫 번째 단계는 "mmap"을 위해서 v4l 디바이스로부터 정보를 얻어 오는 것이다. 관련 소스는 다음과 같다.

```
struct video_mbuf m_buf;
return_value = ioctl(video, VIDIOCGMBUF, &m_buf);
if ( return_value < 0 )
```

```
{
  fprintf(stderr, "%s: Failed to get information of capture memory
  space.₩n", cmd);
  exit(1);
}
```

---

video_mbuf 구조체에 관한 내용은 부록A v4l API를 참조한다. VIDIOCGMBUF ioctl은 "mmap"에 버퍼 크기와 각 프레임을 위한 버퍼 내의 오프셋(offset)에 관한 정보를 알려주는 것이다.

두 번째 단계는 매핑된 메모리 영역(memory mapped area)을 얻어 오는 것이고, 관련 소스는 다음과 같다.

```
void* map;
map = (char*) mmap(0, m_buf.size, PROT_READ, MAP_SHARED, video, 0);
if( map == MAP_FAILED )
{
  fprintf(stderr,"%s: Failed to get pointer to memory mapped
  area.₩n", cmd);
  exit(1);
}
```

세 번째 단계는 매핑된 메모리 영역을 위해 이미지 파라미터(parameters)를 설정하는 것이다. 이 설정 부분도 이미지 포맷이 RGB24일 경우와 YUV420P일 경우가 다르다. 관련 소스는 다음과 같다.

```
struct video_mmap v_map;
v_map.frame = 0;
v_map.height = height;
v_map.width = width;
```

RGB24일 경우: v_map.format = VIDEO_PALETTE_RGB24;

YUV420P일 경우: v_map.format = VIDEO_PALETTE_YUV420P;

9.2.2 "mmap"을 사용하여 프레임 읽어 오기

프레임을 읽어 오기 위해서 VIDIOCMCAPTURE ioctl을 사용하고 관련 소스는 다음과 같다.

```
---------------------------------------------------------------------
return_value = ioctl(video, VIDIOCMCAPTURE, &v_map);
if( return_value < 0 )
{
  fprintf(stderr, "%s: Failed on VIDIOCMCAPTURE ioctl.₩n", cmd);
  exit(1);
}
---------------------------------------------------------------------
```

다음은 VIDIOCSYNC ioctl을 사용하여 프레임의 끝까지 기다린다. 관련 소스는 다음과 같다.

```
---------------------------------------------------------------------
return_value = ioctl(video, VIDIOCSYNC, &v_map);
if( return_value < 0 )
{
  fprintf(stderr, "%s: Failed on VIDIOCSYNC ioctl.₩n", cmd);
  exit(1);
}
---------------------------------------------------------------------
```

9.2.3 매핑 제거

마지막으로 매핑을 제거한다. 이 때 함수 "munmap"을 사용한다. 관련 소스는 다음과 같다.

```
---------------------------------------------------------------------
munmap(map, m_buf.size);
---------------------------------------------------------------------
```

지금까지 이미지를 읽어 오기 위한 두 가지 방법(using read and mmap)에 관해 설명하였다.

10. 이미지 포맷 변환

디바이스가 지원하는 포맷과 저장 포맷의 이미지 포맷이 다를 수 있다. 이 장에서는 두 포맷이 다를 경우에 어느 한쪽으로 동기화 시키기위해 필요한 것들을 다룬다. 위의 두 포맷이 같다면 이 장을 그냥 넘어가도 무방하다.

그럼, 우리가 작성할 프로그램의 경우를 생각해보자. 이제 남은 것은 캡쳐한 이미지를 적당한 파일 포맷으로 저장하는 일이다. 다음 장에서 구체적으로 설명하겠지만 우리가 일차적으로 사용할 이미지 저장 포맷은 ppm이라는 것이다. 이 포맷은 아주 간단한 구조로 되어 있어서 캡쳐한 이미지를 쉽게 저장할 수 있다.

ppm 저장 포맷의 중요한 특징중에 하나는 이미지를 24비트 RGB 이미지 포맷으로 저장한다는 것이다. 앞에서 언급했듯이 입력과 출력의 포맷이 같다면 문제가 없지만, 입력 디바이스가 지원하는 이미지 포맷(보통 YUV 계열)과 저장 포맷의 이미지 포맷이 다를수도 있다.

10.1 내가 사용하는 디바이스는 어떤 이미지 포맷을 지원할까?

그러면 자기가 사용하는 디바이스가 지원하는 이미지 포맷은 어떻게 알 수 있는지 간단한 코드를 사용하여 확인해보자. <linux/videodev.h>에 정의되어 있는VIDIOCGPICT ioctl 콜을 사용하여 확인할 수 있다. 이 부분의 코드는 8장에서 언급한 부분이므로 참고만 하기 바란다.

List: 디바이스의 picture 정보 얻어오기

```
struct video_picture pic;

return_value = ioctl(video, VIDIOCGPICT, &pic);
if ( return_value < 0 )
{
  fprintf(stderr,"%s: Failed to get the image propertiesWn", cmd);
  exit(1);
}
```

VIDIOCGPICT ioctl 콜은 video_picture 구조체를 반환한다. 구조체video_picture의 구조체 중에 palette라는 항목이 있는데 여기에 저장되는 상수값을 확인하여 디바이스가 어떤 이미지 포맷을 지원하는지 확인할 수 있다.

List: "palette" 상수값 확인

```
switch (pic.palette)
{
  case VIDEO_PALETTE_YUV420P:
        printf("YUV 420Wn");
        break;
  case VIDEO_PALETTE_RGB24:
```

```
        printf("24 bit RGB₩n");
        break;
        .
        .
        .
  default:
        break;
}
```

이미지 포맷의 상수값은 <linux/videodev.h>에 잘 나와 있으니 확인하기 바란다.

10.2 이미지 포맷에 대한 이해

영상 관련 장비가 사용하는 이미지 포맷은 크게 YUV 계열과 RGB 계열로 나눌 수 있다. v4l에서 다룰 수 있는 이미지 포맷도 이 두 계열이다.
각각은 다시 데이터의 배열순서나 할당크기에 따라 여러 종류로 나뉜다.

10.2.1 RGB 포맷이란?

RGB는 빨강, 초록, 파랑색의 빛을 한 픽셀에 적절히 배합하여 색상을 만드는 컴퓨터의 화면 표시 원리에 적합하도록 설계된이미지 포맷이다. 각각의 원소에 얼마의 비트를 할당하여 색을 표현하느냐에 따라 16비트, 24비트 등으로 나뉜다. 24비트의 경우 각 색의 원소를 표현하는데 8비트(1바이트)를 할당할 수 있으므로 각 색의 원소를 256단계로 표현할 수 있다.

Figure: 4x2 픽셀 메모리 구조 (24비트 BGR)

```
-------------------------------------------------------------------------
|B00 | G00 | R00 | B01 | G01 | R01 | B02 | G02 | R02 | B03 | G03 | R03|
-------------------------------------------------------------------------
|B10 | G10 | R10 | B11 | G11 | R11 | B12 | G12 | R12 | B13 | G13 | R13|
-------------------------------------------------------------------------
```

위의 그림은 24비트 BGR이 메모리에 저장되는 형식이다. 24비트 BGR은 24비트 RGB와 색상의 배열순서만 다르고 모두 같다. 1개의 픽셀은 3개의 바이트로 표현된다.

10.2.2 YUV 포맷이란?

YUV 포맷은 텔레비전 방송과 관련이 깊다. 텔레비전 방송이 흑백에서 칼라로 바뀌면서 기존의 흑백 방송과 호환을 유지하기 위해 만들어진 포맷이라 할 수 있다. 영상에서 밝기(Y)를 색

상(UV)과 분리하고 따로 저장하여 신호를 송출하면, 칼라로도 볼 수 있을 뿐 아니라 흑백에서 영상을 볼 수 있는 원리를 이용한 것이다. 색상의 경우 파랑은 U, 빨강은 V이다. 초록은 파랑과 빨강을 섞어 만든다.
YUV 포맷은 명도와 채도의 할당 비율에 따라 4:2:2, 4:2:0, 4:1:1 등으로 나뉜다. YCbCr이라는 형식도 있는데 YUV가 변형된 형태이다. Cb는 파랑(U), Cr은 빨강(V)이다.

Figure: 4x2 픽셀 메모리 구조 (YUV 420)

```
-----------------------
|Y00 | Y01 | Y02 | Y03|
-----------------------
|Y10 | Y11 | Y12 | Y13|
-----------------------
|U00 | U02|
-----------
|V00 | V02|
-----------
```

위 그림은 4:2:0의 비율의 YUV 메모리 구조이다. Y는 1픽셀에 1바이트 씩 할당하고, U와 V는 각각 4픽셀에 1바이트씩 할당한다.

10.3 YUV -> RGB 변환

디바이스가 YUV만 지원한다고 확인된 경우, RGB로 변환해야 한다. 여기서는 변환 함수를 만들어 그것을 사용한다. 함수 원형은 다음과 같다.

List: YUV -> RGB 변환 함수 원형

```
-------------------------------------------------------------------------
static void yuv2rgb_init(void);
static void yuv2rgb (char *out_addr, char *in_addr, int rowstride,
                     int width,   int height);
-------------------------------------------------------------------------
```

두개의 함수가 필요하다.
- yuv2rgb_init(): RGB로 변환시 변환 공식에 따라 그때 그때 계산하는 것이 아니라, 대응되는 값들을 미리 계산해 놓고 참조만으로 변환이 가능하게 만든다.
- yuv2rgb(): 캡쳐한 YUV 포맷의 이미지를 새로운 메모리에 RGB 형식으로 저장한다. out_addr는 캡쳐한 이미지를 RGB로 저장하기 위해 잡아놓은 메모리의 포인터이고, in_addr는 캡쳐한 YUV 원본이 저장되어 있는 메모리의 포인터이다.

그럼 이 두 함수를 어떻게 사용하는지 알아보자. 변환을 위해서 우리는 malloc()을 사용하여 새로운 메모리를 하나 잡아야한다. 그 다음 그 메모리에 위의 두 함수를 사용하여 이미지 포맷 변환을 수행한다.

List: YUV -> RGB 변환 함수 쓰기

```
----------------------------------------------------------------------
image_buffer = (char*) malloc(width * height * depth / 8);
yuv2rgb_init();
yuv2rgb(image_buffer, map, width * 3, width , height);
----------------------------------------------------------------------
```

malloc으로 잡는 메모리의 크기는 캡쳐하고자 하는 이미지의 픽셀수(높이x폭)에 한개의 픽셀에 필요한 바이트수를 곱하여 구한다.

YUV 포맷에서 RGB 포맷으로 바꾸는 구체적인 알고리즘은 두 이미지 포맷의 스펙(Spec.)을 정확히 이해하고 있어야 가능한 일이다. 여기서는 구체적인 변환 기술에 대해서는 언급하지 않고, 대략의 변환과정을 기존의 알고리즘을 통해 설명하고자 한다.

변환의 개략적인 원리는 다음과 같다.
YUV는 색상 정보가 파랑과 빨강 밖에없기 때문에 RGB의 초록에 대응되는 색상을 따로 만들어야 한다. 거기다 분리되어 있는 밝기와 색상 정보를 합쳐야 한다. 또 변환 과정중에 RGB의 결과값이 0과 255사이를 넘어서 이미지가 올바르게 나오지 않게 되는데, 그런 값들은 적당한 보정 작업을 거쳐야한다.

List: 함수 yuv2rgb_init(), yuv2rgb()

```
----------------------------------------------------------------------
/* header */
#define CLIP            320
#define RED_NULL      128
#define BLUE_NULL     128
#define LUN_MUL       256
#define RED_MUL       512
#define BLUE_MUL      512

#define GREEN1_MUL   (-RED_MUL/2)
#define GREEN2_MUL   (-BLUE_MUL/6)
#define RED_ADD       (-RED_NULL   * RED_MUL)
#define BLUE_ADD      (-BLUE_NULL * BLUE_MUL)
#define GREEN1_ADD   (-RED_ADD/2)
```

```
#define GREEN2_ADD   (-BLUE_ADD/6)

/* lookup tables */
static unsigned int   ng_yuv_gray[256];
static unsigned int   ng_yuv_red[256];
static unsigned int   ng_yuv_blue[256];
static unsigned int   ng_yuv_g1[256];
static unsigned int   ng_yuv_g2[256];
static unsigned int   ng_clip[256 + 2 * CLIP];      // 칼라 보정을 위해

#define GRAY(val)                 ng_yuv_gray[val]
#define RED(gray,red)             ng_clip[ CLIP + gray + ng_yuv_red[red] ]
#define GREEN(gray,red,blue)      ng_clip[ CLIP + gray + ng_yuv_g1[red] + ₩
                                                  ng_yuv_g2[blue] ]
#define BLUE(gray,blue)           ng_clip[ CLIP + gray + ng_yuv_blue[blue] ]

static void yuv2rgb_init(void)
{
  int i;

  /* init Lookup tables */
  for ( i = 0; i < 256; i++ )
  {
    ng_yuv_gray[i] = i * LUN_MUL >> 8;
    ng_yuv_red[i]   = (RED_ADD      + i * RED_MUL)      >> 8;
    ng_yuv_blue[i] = (BLUE_ADD     + i * BLUE_MUL)     >> 8;
    ng_yuv_g1[i]    = (GREEN1_ADD + i * GREEN1_MUL) >> 8;
    ng_yuv_g2[i]    = (GREEN2_ADD + i * GREEN2_MUL) >> 8;
  }
  for ( i = 0; i < CLIP; i++ )
    ng_clip[i] = 0;
  for ( ; i < CLIP + 256; i++ )
    ng_clip[i] = i - CLIP;
  for ( ; i < 2 * CLIP + 256; i++ )
    ng_clip[i] = 255;
}

static void
yuv2rgb(char *out_addr, char *in_addr, int rowstride, int width, int
```

```
height)
{
  unsigned char *y,*u,*v;
  unsigned char *us,*vs;
  unsigned char *dp,*d;
  int i,j,gray;

  dp = (unsigned char *)out_addr;
  y  = (unsigned char *)in_addr;
  u  = y +  width * height;
  v  = u +  width * height / 4;

  for ( i = 0; i < height; i++ )
  {
    d = dp;
    us = u; vs = v;
    for ( j = 0; j < width; j += 2 )
    {
      gray    = GRAY(*y);
      *(d++) = BLUE(gray,*u);
      *(d++) = GREEN(gray,*v,*u);
      *(d++) = RED(gray,*v);
      y++;
      gray    = GRAY(*y);
      *(d++) = BLUE(gray,*u);
      *(d++) = GREEN(gray,*v,*u);
      *(d++) = RED(gray,*v);
      y++; u++; v++;
    }
    if ( 0 == (i % 2) )
    {
      u = us; v = vs;
    }
    dp += rowstride;
  }
}
```

헤더 파일에 들어갈 상수값들과 매크로는 변환 공식을 표현한 것이다.

11. PPM으로 저장하기

우리는 앞서 카메라를 통해 정지화상을 캡쳐하였으며, 이를 RGB 포맷으로 알맞게 변환하는 과정을 살펴 보았다. 이제 이렇게 얻어진 데이터를 디스크에 저장하여 영구히 보존할 수 있는 방법을 알아보자.

이미지를 디스크에 저장하고자 할 경우에 가장 먼저 고려해야 할 사항은 '어떠한 형태로 이미지가 저장 될 것인가' 이다. 여기서 언급한 이미지의 형태는 기본적으로 저장되는 파일의 형태를 뜻하며, 우리는 다소 단순한 형태의 이미지 파일 포맷인 PPM 파일을 사용 할 것이다.

PPM 파일은 원본 이미지를 손실 시키지 않고 RGB 형태로 그대로 저장 할 수 있는 트루 칼라(True Color) 형태의 이미지 파일 포맷을 말한다.

앞 장에서 우리는 입력 받은 이미지를 저장하기 쉽게 하기 위해서 RGB 포맷을 갖도록 수정하였다. 따라서 이러한 Raw 데이타는 PPM 포맷을 갖는 이미지 파일로 다음과 같이 쉽게 저장할 수 있다.

Example: Save a still image to PPM File

```
01: FILE *fin;
02:
03: fin = fopen(file, "w");
04: if ( NULL == fin )
05: {
06:     fprintf(stderr, "file %s could not be openedWn", file);
07:     exit(1);
08: }
09:
10: fprintf(fin, "P6 # P6 TypeWn");
11: fprintf(fin, "# Capture a still image by ELFWn");
12: fprintf(fin, "%d # Width is %d pixelsWn", width);
13: fprintf(fin, "%d # Height is %d pixelsWn", height);
14: fprintf(fin, "%d # True ColorWn", (int) pow(2, (depth/3))-1);
15:
16: fwrite(image_buffer, width * height * depth, 1, fin);
17:
18: fclose(fin);
```

위의 예제에서는 FILE 구조체의 포인터 fin을 이용해 파일 명령행 인수로 입력 받은 파일명을 이용하여 쓰기 위한 파일을 3번째 라인(Line)에서 열고 있다. 그리고 10번째 라인 부터가 실질적인 PPM 포맷의 내용이다. PPM 포맷은 P3 또는 P6의 두가지 형태를 갖는다. 현재 파일은 P6로 P3에 비해 몇가지 장점이 있다. 헤더라 불리우는 10번 부터 14번째 라인 까지의 내용은 저장 될 이미지 파일의 정보를 간단히 담고 있다. 첫번째 라인에는 현재 PPM의 타입을 기술하며, 이어서 이미지의 폭과 높이를 픽셀로 나타낸 후 이미지의 각 픽셀의 RGB 요소 중 한가지 요소가 갖을 수 있는 최대 값을 명시한다. 기본적으로 '#'으로 시작되는 문장은 주석이며 해당 라인의 끝까지를 주석으로 해석한다.
헤더를 모두 파일로 출력한 후 실질적인 데이타를 출력 하게 되는데, P6의 형태에서는 RGB 포맷의 Raw 데이타를 공백(White space) 없이 그대로 출력 하면 된다(16번째 라인). 그러나 P3 형식을 갖는 파일에서는 데이타를 문자 그대로가 아닌 10진수 숫자로 변환하여 ASCII로 해당 숫자를 기록한다. 또한 P3 형태의 파일은 한 라인에 70 컬럼을 넘을 수 없다는 제약을 갖는다. 기본적으로 PPM 형태의 파일은 각 픽셀의 RGB 요소 각각을 공백으로 구분하여 저장하게 된다.

저장이 완료된 PPM형태의 P3 파일을 다음과 같을 수 있다.

Example: PPM File

```
-----------------------------------------------------------------------
P3 # P3 형태
# ELF 정지영상 캡쳐하기
640 # 폭은 640 픽셀
480 # 높이는 480 픽셀
255 # 트루 칼라
0 0 0   0 0 0   0 0 0   0 0 0   0 0 0   0 0 0   0 0 0   0 0 0   0 0 0
0 0 0   0 0 0   0 0 0   0 0 0   0 0 0   0 0 0   0 0 0   0 0 0   0 0 0
0 0 0   0 0 0   0 0 0   0 0 0   0 0 0   0 0 0   0 0 0   0 0 0   0 0 0
...
-----------------------------------------------------------------------
```

위의 예제에서 첫번째 즐(Line)은 PNM 파일의 타입을 말하며, 640 x 480 픽셀을 사용한다. 또한 한 픽셀의 각 칼라 컴포넌트(Color Component)는 최대 255의 값을 갖을 수 있음을 알 수 있다. 데이타 부분은 모든 RGB요소가 0으로 검은색으로 보이는 픽셀이 한 라인 마다 9개씩 있으므로 적어도 이미지 파일의 처음 27픽셀(Pixels)은 검은색으로 표현된다.

12. 정리하기

12.1 파일 닫기

마지막으로 프로그램을 종료하기 전에 프로그램과 디바이스와의 연결을 해제해주자. 방법은 매우 간단하다. close 시스템 콜을 사용해서 디바이스 파일을 닫는다. close함수의 인수는 디바이스 파일 기술자만 있으면 된다.

```
-------------------------------------------------------------------------
#include<fcntl.h>
-------------------------------------------------------------------------
close(video);
-------------------------------------------------------------------------
```

PPM 이미지로 저장하기 위해 사용한 파일 포인터도 닫아줘야 한다. 프로그램에서 데이터가 완전히 기록되도록 하기 위해서 fclose를 호출하자. 이것도 close와 마찬가지로 파일 기술자만 인수로 주면 된다.
사용 예는 위 11절에 있으니 참조하자.

12.2 메모리 해제

malloc으로 할당받은 메모리는 free함수를 사용해서 시스템으로 되돌려 줘야 한다. free로 메모리를 해제해 주는 것은 각 장에서 설명했다.

12.3 종료하기

close를 해서 디바이스 파일을 닫고, 메모리도 해제한 후에 프로그램을 종료하면 된다.

```
-------------------------------------------------------------------------
/* 우리 프로그램에서 main이 integer반환값을 가지므로 성공적인 종료를
의미하는 EXIT_SUCCESS(0)을 반환한다.(EXIT_SUCCESS는 stdlib.h에 정의) */
return EXIT_SUCCESS;
-------------------------------------------------------------------------
```

13. 참고문서

[1] Ben Salama, UNIX System Programming 2/e, Addison-Wesley, 1999.
[2] frame buffer 이야기(4), http://kelp.or.kr/korweblog/stories.php?story=02/11/11/1742716
[3] man page와 info page.
[4] Paul Bourke, PPM / PGM / PBM image files, http://astronomy.swin.edu.au/~pbourke/dataformats/ppm/, 1997.
[5] simple image grabber, http://pl.smu.ac.kr/lxr/simple-image-grabbler/source

[6] v4l kernel Doc API, http://www.unix-systems.org/single_unix_specification
[7] VFL How to, http://pages.cpsc.ucalgary.ca/~sayles/VFL_HowTo
[8] Video Codecs and Pixel Formats, www.fourcc.org
[9] Video for Linux Two - Image Data Formats, http://www.thedirks.org/v4l2/v4l2fmt.htm
[10] W. Richard Stevens, "Advanced Programming in the UNIX Environment", ADDISON-WESLEY, 1992.
[11] 권태완, Digital Image Processing, http://mcl.hallym.ac.kr/twkwon/2_lecture/Digital_Image_Processing, 2003.
[12] 이만재, 홍명희, 박현재, "멀티미디어 개론", 한국방송대학교출판부, 1999.
[13] 이필두, "디지틀 영상 이론에서 활용까지-Premiere & After Effects",(주)영진닷컴, 2002.

부록 A v4l API

v4l에서 사용되는 구조체는 /usr/include/linux/videodev.h 파일에 정의되어 있다. 이 API 정리는 간단하게 이미지를 캡쳐하는 것에 관련된 부분만 정리했음을 미리 알린다.

A.1 video_capability

비디오 캡쳐 디바이스(Device)가 지원하는 기능에 관한 정보를 얻으려면VIDIOCGCAP ioctl을 이용한다. ioctl에 struct video_capability를 건네주면 결과값을 돌려준다. struct video_capability는 다음과 같다.

```
struct video_capability
{
  char name[32];   /* Canonical name for this interface */
  int type;          /* Type of interface */
  int channels;    /* Num channels */
  int audios;       /* Num audio devices */
  int maxwidth;    /* Supported width */
  int maxheight;   /* And height */
  int minwidth;    /* Supported width */
  int minheight;   /* And height */
};
```

- name: 회사에서 제공하는 캡쳐 디바이스의 정식 이름을 나타낸다. 이 name의 type은 char 배열이다.

- type: 캡쳐 디바이스가 제공하는 type의 정보를 알 수 있다. type은 int형으로 & 연산을 통

해 디바이스가 제공하는 기능들을 검출해 내기
때문에 type의 값을 보면 디바이스가 제공하는 기능들을 알아낼 수 있다. 예를 들어, 디바이스가 캡쳐(1)와 튜닝(tuning)기능(2)을 제공한다면 type에는 3이라는 값이 저장된다.

```
VID_TYPE_CAPTURE        1      /* 캡쳐를 할 수 있다 */
VID_TYPE_TUNER          2      /* 조정을 할 수 있다 */
VID_TYPE_TELETEXT       4      /* 텔레텍스트를 제공한다 */
VID_TYPE_OVERLAY        8      /* 프레임 버퍼에 입힌다 */
VID_TYPE_CHROMAKEY      16     /* chromakey에 의해 덧 씌운다*/
VID_TYPE_CLIPPING       32     /* 잘라내기가 가능한다 */
VID_TYPE_FRAMERAM       64     /* 프레임 버퍼 메모리를 사용한다 */
VID_TYPE_SCALES         128    /* 이미지 크기 조절이 가능하다 */
VID_TYPE_MONOCHROME     256    /* 흑백만 가능 */
VID_TYPE_SUBCAPTURE     512    /* 이미지의 일부분을 캡쳐할 수 있다 */
VID_TYPE_MPEG_DECODER   1024 /* MPEG streams을 해제할 수 있다 */
VID_TYPE_MPEG_ENCODER   2048 /* MPEG streams을 암호화할 수 있다 */
VID_TYPE_MJPEG_DECODER 4096 /* MJPEG streams을 해제할 수 있다 */
VID_TYPE_MJPEG_ENCODER 8192 /* MJPEG streams을 암호화할 수 있다 */
```

- channels: int형으로 현재 사용 가능한 텔레비전/오디오의 채널의 개수를 나타낸다.
- audios: int형으로 현재 사용 가능한 오디오(audio) 디바이스의 개수를 나타낸다.
- maxwidth: int형으로 디바이스가 캡쳐할 수 있는 최대폭을 픽셀 단위로 나타낸다.
- maxheight: int형으로 디바이스가 캡쳐할 수 있는 최대높이를 픽셀 단위로 나타낸다.
- minwidth: int형으로 디바이스가 캡쳐할 수 있는 최소폭을 픽셀 단위로 나타낸다.
- minheight: int형으로 디바이스가 캡쳐할 수 있는 최소높이를 픽셀 단위로 나타낸다.

예1) capture.c

```
#include <sys/types.h>          /* for using open */
#include <sys/ioctl.h>          /* for using ioctl */
#include <linux/videodev.h>   /* for using v4l API */
#include <fcntl.h>

int main()
{
  struct video_capability cap;
  int fd, ret;
```

```
  if ( (fd = open("/dev/video0", O_RDONLY)) < 0 )
  {
    printf("Opening video0 failed.₩n");
    exit(1);
  }

  if ( (ret = ioctl(fd, VIDIOCGCAP, &cap)) < 0 )
  {
    printf("VIDIOCGCAPTURE failed₩n");
    exit(1);
  }

  printf("The name is %s₩n", cap.name);
  printf("The type is %d₩n", cap.type);
  printf("The number of channels is %d₩n", cap.channels);
  printf("The number of audio devices is %d₩n", cap.audios);
  printf("The maxwidth is %d₩n", cap.maxwidth);
  printf("The maxheight is %d₩n", cap.maxheight);
  printf("The minwidth is %d₩n", cap.minwidth);
  printf("The minheight is %d₩n", cap.minheight);

  return 0;
}
```

결과1)

```
The name is OV511+ USB Camera <-- "OV511+ USB Camera"가 이름이다.
The type is 513 <-- Capture(1) + Subcapture(512)가 가능하다
The number of channels is 1 <-- 이 디바이스가 지원하는 채널은 1개 이다.
The number of audio devices is 0 <-- 오디오 디바이스는 없다.
The maxwidth is 640 <-- 캡쳐가 가능한 최대폭은 640 픽셀이다.
The maxheight is 480 <-- 캡쳐가 가능한 최대높이는 480 픽셀이다.
The minwidth is 64 <-- 캡쳐가 가능한 최소폭은 64 픽셀이다.
The minheight is 48 <-- 캡쳐가 가능한 최소높이는 48 픽셀이다.
```

A.2 Video Sources

각각의 video4linux 비디오/오디오 디바이스들은 1개 이상의 채널을 가질수 있다. 각각의 채널

은 VIDIOCGCHAN ioctl을 이용해서 struct video_channel을 넘겨주면 결과값으로 각 채널의 정보를 돌려준다. 우선 이 ioctl을 사용하기 전에 사용자는 먼저 사용할 채널을 선택해야 한다. 또한 VIDIOCSCHAN ioctl은 정수 인수를 입력받아 그 채널로 입력을 바꿀 수 있도록 한다. 하지만 색이나 튜닝에 대한 설정은 채널을 옮기면 다시 설정해야 한다.
struct video_channel의 구조는 다음과 같다.

```
struct video_channel
{
  int channel;       /* The channel number */
  char name[32];    /* The input name - preferably reflecting the
                          label on the card input itself */
  int tuners;        /* Number of tuners for this input */
  __u32   flags;     /* Properties the tuner has */
  __u16   type;      /* Input type (if known) */
  __u16   norm;      /* Norm set by channel */
};
```

- channel: int형으로 사용하고자 하는 채널의 번호를 나타낸다.
- name: char 배열형으로 입력의 이름이다. 입력 이름은 입력 자체로 카드의 레이블을 반영하는 것이 좋다.
- tuners: int형으로 선택한 입력에서 제공하는 튜너의 개수를 나타낸다.
- flags: 32비트 unsigned int형으로 튜너가 가지고 있는 특성을 나타낸다.

```
VIDEO_VC_TUNER      1     /* 채널이 튜너를 가지고 있다 */
VIDEO_VC_AUDIO      2     /* 채널이 오디오를 가지고 있다 */
```

- type: 16비트 unsigned short형으로 입력 형태를 나타낸다.

```
VIDEO_TYPE_TV       1     /* 입력이 TV 입력이다 */
VIDEO_TYPE_CAMERA 2     /* 입력이 카메라입력이다 */
```

- norm: 16비트 unsigned short형으로 선택한 채널의 표준
(텔레비전 신호의 모드)의 설정을 나타낸다.

A.3 Capture Window

캡쳐되는 영역은 struct video_window에 의해서 정의된다. 이것은 캡쳐 영역과 잘라내기 정보를 정의한다. VIDIOCGWIN ioctl은 결과값으로 현재 설정에 대한 정보를 반환한다. VIDIOCSWIN ioctl은 새로운 값으로 설정하게 된다. 하지만 VIDIOCSWIN ioctl을 사용했다고 해서 원하는 값으로 설정됐다고 확신할 수는 없다. 왜냐하면 디바이스가 사용자가 원하는 설정을 지원하지 못할 수 도 있기 때문이다. 따라서 VIDIOCSWIN ioctl을 사용하기 전에 VIDIOCGWIN ioctl을 통해서 적합한 설정의 범주를 알아야 한다.

struct video_window의 구조는 다음과 같다.

```
struct video_window
{
  __u32     x,y;                        /* Position of window */
  __u32     width,height;               /* Its size */
  __u32     chromakey;
  __u32     flags;
  struct    video_clip *clips;          /* Set only */
  int       clipcount;                  /* Set only */
};
```

- x: 32비트 unsigned int형으로 X윈도우에서의 x좌표를 나타낸다.
- y: 32비트 unsigned int형으로 X윈도우에서의 y좌표를 나타낸다.
- width: 32비트 unsigned int형으로 캡쳐하는 이미지의 폭을 나타낸다.
- height: 32비트 unsigned int형으로 캡쳐하는 이미지의 높이를 나타낸다.
- chromakey: 32비트 unsigned int형으로 크로마 키 값을 나타낸다.
- flags: 32비트 unsigned int형으로 추가적인 캡쳐 플래그 값을 나타낸다.
- struct video_clip *clips: 클리핑 사각형의 리스트를 나타낸다. (설정만 가능하다) 이곳에서 사용되는 struct video_clip의 구조는 다음과 같다.

```
struct video_clip
{
  __s32     x,y;
  __s32     width, height;
  struct    video_clip *next; /* For user use/driver use only */
};
```

. x: 32비트 signed int형으로, 스킵하는 사각형의 x 좌표를 나타낸다.
. y: 32비트 signed int형으로, 스킵하는 사각형의 y 좌표를 나타낸다.
. width: 32비트 signed int형으로, 스킵하는 사각형의 폭을 나타낸다.
. height: 32비트 signed int형으로, 스킵하는 사각형의 높이를 나타낸다.
. next: 다음의 clip을 가리키게 된다.

- clipcount: int형으로 클리핑 사각형의 개수를 나타낸다. (설정만 가능하다.)

A.3.1 부연 설명

부수적으로 캡쳐 디바이스가 Subcapture기능을 제공한다면(즉, capability의 type중에 VID_TYPE_SUBCAPTURE가 설정되어 있다면 = type의 값이 512 + a라면) video_capture라는 구조를 이용하여 캡쳐 영역의 일부만을 캡쳐할 수 있게 된다. struct video_capture라는 구조체를 이용한다. 이 구조체는 캡쳐할 시간과 캡쳐할 특정 부분을 지정한다.
이 구조체의 구성은 다음과 같다.

```
struct video_capture
{
  __u32    x,y;                 /* Offsets into image */
  __u32    width, height;       /* Area to capture */
  __u16    decimation;          /* Decimation divder */
  __u16    flags;               /* Flags for capture */
};
```

- x: 32비트 unsigned int형으로 일부분 캡쳐할 사각형의 x 좌표를 나타낸다.
- y: 32비트 unsigned int형으로 일부분 캡쳐할 사각형의 y 좌표를 나타낸다.
- width: 32비트 unsigned int형으로 일부분 캡쳐할 사각형의 폭을 나타낸다.
- height: 32비트 unsigned int형으로 일부분 캡쳐할 사각형의 높이를 나타낸다.
- decimation: decimation은 16비트 unsigned short형으로 적용될 스킵할 영역을 나타낸다.
- flags: 16비트 unsigned short형으로 일부분 캡쳐할 때의 설정을 나타낸다.

```
VIDEO_CAPTURE_ODD          0      /* 홀수 프레임만 캡쳐한다 */
VIDEO_CAPTURE_EVEN         1      /* 짝수 프레임만 캡쳐한다 */
```

* 어떤 역자에 의하면 요즘에는 video_capture 구조체 및, VIDIOCGCAPTURE VIDIOCSCAPTURE ioctl는 사용되고 있지 않다.

A.4 Image Properties

그림의 이미지의 특성은 VIDIOCGPICT ioctl을 이용하여 struct video_picture에서 알아낼 수 있다. VIDIOCSPICT를 이용하면 새로운 값들을 설정할 수 있다. palette type을 제외하고는 모든 값이 0-65535의 값을 취할 수 있다.

```
struct video_picture
{
  __u16    brightness;      /* Picture brightness */
  __u16    hue;             /* Picture hue(colour only) */
  __u16    colour;          /* Picture colour(colour only) */
  __u16    contrast;        /* Picture contrast */
```

```
  __u16    whiteness;       /* Black and white only */
  __u16    depth;           /* Capture depth */
  __u16    palette;         /* Palette in use */
};
```

- brightness: 16비트 unsigned short형으로 그림의 밝기를 나타낸다. 큰 값을 설정하면 밝은 그림이 된다.
- hue: 16비트 unsigned short형으로 그림의 색조를 나타낸다. 칼라에서만 가능하다.
- colour: 16비트 unsigned short형으로 그림의 색을 나타낸다. 칼라에서만 가능하다.
- contrast: 16비트 unsigned short형으로 그림의 대조를 나타낸다.
- whiteness: 16비트 unsigned short형으로 그림의 백색도를 나타낸다. 흑백에서만 가능하다.
- depth: 16비트 unsigned short형으로 그림의 캡쳐 깊이를 나타낸다. 프레임 버퍼의 깊이와 같게 설정할 필요가 있다.
- palette: 16비트 unsigned short형으로 현재 설정을 하고자 하는 그림의 팔레트를 나타낸다.

```
VIDEO_PALETTE_GREY      1       /* 선형적으로 증가하는 gray
                                   scale(255가 가장 밝은 흰색) */
VIDEO_PALETTE_HI240     2       /* BT848의 8bit칼라 큐브 */
VIDEO_PALETTE_RGB565    3       /* 565 16bit RGB */
VIDEO_PALETTE_RGB24     4       /* 24bit RGB */
VIDEO_PALETTE_RGB32     5       /* 32bit RGB */
VIDEO_PALETTE_RGB555    6       /* 555 15bit RGB */
VIDEO_PALETTE_YUV422    7       /* YUV422 캡쳐 */
VIDEO_PALETTE_YUYV      8
VIDEO_PALETTE_UYVY      9       /* The great thing about standards is ... */
VIDEO_PALETTE_YUV420    10
VIDEO_PALETTE_YUV411    11      /* YUV411 캡쳐 */
VIDEO_PALETTE_RAW       12      /* RAW capture (BT848) */
VIDEO_PALETTE_YUV422P   13      /* YUV 4:2:2 Planar */
VIDEO_PALETTE_YUV411P   14      /* YUV 4:1:1 Planar */
VIDEO_PALETTE_YUV420P   15      /* YUV 4:2:0 Planar */
VIDEO_PALETTE_YUV410P   16      /* YUV 4:1:0 Planar */
VIDEO_PALETTE_PLANAR    13      /* start of planar entries */
VIDEO_PALETTE_COMPONENT 7       /* start of component entries */
```

A.5 Frame buffer

캡쳐 카드로 부터 프레임 버퍼(frame buffer)로 데이터를 직접 저장하기 위해서는 프레임 버퍼의 베이스 주소(base address), 크기(size)와 구조(Organisation)를 디바이스 드라이버에게 알려 주어야 한다.

```
struct video_buffer
{
  void      *base;              /* base address */
  int       height,width;     /* height, width of the frame buffer        */
  int       depth;              /* depth of the frame buffer */
  int       bytesperline;     /* Number of bytes of memory between
                                    the start of two adjacent lines */
};
```

- base: 버퍼의 베이스 물리적 주소를 나타낸다. 이것을 NULL로 설정하면 물리적으로 프레임 버퍼에 접근하지 않겠다는 것을 의미한다.
- height: int형으로 프레임 버퍼의 높이를 나타낸다.
- width: int형으로 프레임 버퍼의 폭을 나타낸다.
- depth: int형으로 프레임 버퍼의 깊이를 나타낸다.
- bytesperline: int형으로 근접한 라인이 시작되는 부분까지의 메모리의 크기를 나타낸다.

A.6 Reading Images

'read' 시스템 콜을 호출할 때마다 디바이스로부터 가능한 다음 이미지를 가져오게 된다. VIDIOCSPICT와 VIDIOCSWIN을 이용하여 포멧과 크기를 설정하는 것은 사용자(호출자)의 책임이다. 이미지를 캡쳐를 처리하는 또 다른 방법은 디바이스가 지원만 한다면 mmap 인터페이스를 사용할 수도 있다. 우선 mmap을 사용하기 위해선 사용자는 이미지의 크기와 깊이를 설정해야 하고 VIDIOCGMBUF ioctl을 사용해야 한다. 이 ioctl은, mmap 해야 할 버퍼(Buffer)의 크기와 각각의 프레임(Frame)에 대한 버퍼에서의 오프셋(offset)을 알려준다. 지원되는 프레임의 수는 디바이스에 의해서 결정되는데 대체적으로 한 프레임이다.

struct video_mbuf의 구조는 다음과 같다.

```
struct video_mbuf
{
   int    size;              /* Total memory to map */
   int    frames;             /* Frames */
   int    offsets[32];
};
```

- size: int형으로 map하고자 하는 메모리의 총양을 나타낸다. 단위는 byte이다.
- frames: int형으로 프레임의 개수를 나타낸다.
- offsets: int형으로 각 프레임의 오프셋을 나타낸다. 기본적으로는 videodev.h파일에는 VIDEO_MAX_FRAME을 32로 정의했기 때문에 위에서 int offsets[32]라고 표현했다.

mmap은 VIDIOCMCAPTURE ioctl을 이용하여 video_mmap 구조에 저장해 높은 포맷과 이미지의 크기를 이용하여(최초의 설정과 같거나 작을 수 있다.) 프레임으로 캡쳐를 시작하라고 시킬 수 있다. 만약에 VIDIOCMCAPTURE ioctl이 아직 캡쳐를 하지 않았다는 결과를 반환하면, 드라이버는 하드웨어에게 프레임을 캡쳐하라고 명령한다. VIDIOCMCAPTURE ioctl이 사용된 후에는 VIDIOCSYNC ioctl을 이용하여 프레임이 완전하게 캡쳐가 될 때까지 기다리도록 한다. VIDIOCCSYNC는 캡쳐를 완료하고 싶은 프레임 번호를 인수로 가진다. v4l에서는 프레임을 최대 32개 까지 사용할 수 있다.

부록 B Terms

B.1 서브샘플링(Subsampling)

서브샘플링이라 하면 주어진 정보의 일부분만을 사용하는 것을 말한다. 사진을 디지틀화 할 경우 매 픽셀은 RGB의 세 가지 성분으로 표시된다. 그러나 같은 색을 표시하는 경우 RGB대신 YUV로 표시할 수 있다. YUV에서 Y는 밝기를 나타내고 U와 V는 색사정보만을 포함하고 있다. 사람의 눈은 밝기에는 민감하나 색상정보에는 상대적으로 둔감하다. 따라서 Y 정보를 4번 사용할 때 U나 V 정보를 2번 사용하면 전체적으로 정보량은 2/3으로 줄어들게 된다.이 경우 4:2:2 서브샘플링이라 한다.
서브샘플링은 다른 형태로도 사용된다.RGB를 각각 8bit씩 총 3byte를 사용하는 대신 5bit씩 사용하여 2byte만을 사용하면 역시 2/3로 압축할 수 있다. 이러한 방법을 사용하면 적,녹,청색의 각 색조를 256단계로 보여주는 대신 32단계로 보여주는 것으로, 자세히 관찰하면 차이가 나타나나 일반 목적으로 사용할 경우 별로 압축 하였다는 느낌을 주지 않는다.

B.2 TV 방식

양대 대전을 치르며 각국은 동맹국별로 일정한 호환성을 유지하는 서로 다른 TV방송 표준을 개발, 호환 불가능한 독자적 표준을 사용하기에 이른다. 크게 대별해 보면 미국의 영향권에서는 NTSC방식이, 옛 서유럽 국가에서는 PAL방식이, 프랑스와 옛 동유럽지역에서는 SECAM방식이 사용되고 있다.

B.2.1 초당 프레임수

방식	프레임수
NTSC	29.97
PAL, SECAM비디오	25
영화	24

B.2.2 NTSC(National Television Standard Committee)

NTSC방식은 1953년 미국에서 칼라 TV표준 방식으로 채택하였으며, 일본에서도 1960년 6월에 표준 방식으로 채용하였다. 국내에서는 1980년에 칼라 방식의 방송 도입 문제로 각계에서 NTSC, PAL등의 송출 방식에 대해 여러 의견이 있었으나 주무 관청인 전파관리국은 이미 할당된 채널 기준과 운용되고 있는 방송 장비의 특성을 고려하여 NTSC방식을 채택하였다. CCTV시스템에서는 일반적으로 NTSC방식을 주로 채택하고 있다.이 방식은 미국의 표준 방식으로 주사선 525개를 사용한다. 세계적으로 미국 및 북미지역과 일본, 우리나라 등 약 25개국에서 사용하는 방식이다.

B.2.3 PAL(Phase Alternation by Line)

PAL방식은 1962년 독일의 Telefunken사가 개발한 칼라 TV방식으로 NTSC방식의 결점인 위상왜곡(Phase Distortion)이 개선되었다. NTSC와 가장 다른 점은 2개의 색신호 중에 한쪽의 위상을 주사선마다 반전시키고 있다는 것이다. 즉, 'B-Y' 신호는 90도의 위상으로 고정, 'R-Y' 신호의 쪽은 주사선마다 0도와 180도로 위상을 반전해서 보낸다. 이와 같이 하면 전송 시스템에서 색 신호의 위상 왜곡은 주사선마다 상쇄되기 때문에 위상 관리는 NTSC에 비해 용이하게 된다. 즉, 색상의 변화가 없다는 것으로 NTSC방식의 결점인 장거리 중계 회선 중에서 생기는 색상 일그러짐의 영향이 경감된다. 단, 색부반송파가 시간적으로 연달아 2라인에 걸쳐 평균되기 때문에 색의 수직 방향의 해상도가 저하된다.
서독이 개발하여 영국 등 서유럽 지역과 동남아, 아프리카 등 48개국에서 사용하고 있는 방식으로 NTSC의 단점인 수신 지역에 따른 색상의 일그러짐을 극복하였고 주사선 625개를 사용하여 일반적으로 NTSC보다 좋은 화질을 구현한다.

B.2.4 SECAM(Sequential Couleur A Memoire)

SECAM은 1953년 프랑스의 Henriderk가 제안한 칼라 TV방식으로 1967년 프랑스에서 처음으로 방송되었다. SECAM방식이란 색 신호를 라인 순차로 전송하여 메모리 기술로 복원하는(sequential couleur a memoire)것이다. 두 개의 색차 신호를 NTSC나 PAL방식과 같이 동시에 보내지 않고, 선 순차 방식으로 색부반송파를 FM 변조하여 전송한 다음 수신 측에서 1라인 메모리를 사용하여 동시 신호로 변환한다.
이 방식은 화상이 안정되어 있고 전송 경로에서 일어나는 왜곡에 강한 장점이 있다. 즉, 전송계에서 생기는 일그러지는 영향은 적게 받지만 색상이 변하지 않고 수상기 색조정의 필요가 없다. 그러나 무채색에 대해서도 세부반송파의 진폭이 0이 되지 않기 때문에 흑백 TV로 의 시청이 불가능하고 색부반송파에 의한 방해(점상 노이즈)가 눈에 띄기 쉽고 수상기의 가격이 타 방식보다 비싸다는 단점이 있다. SECAM은 주사선 819개로 프랑스가 개발하여 프랑스 언어권의 아프리카 지역과 러시아 등 동구권에서 채택하고 있다.

B.3 팔레트

픽셀은 'Picture Element'의 합성어로 모니터 화면을 구성하는 가장 기본이 되는 단위를 말한

다. 즉, 일반적으로 모니터를 통해서 보여 지는 이미지는 작은 사각형의 점, 즉 픽셀들로 구성되어있다. 따라서 이러한 이미지는 픽셀을 기본 단위로 하는 집합으로 표시되며 이렇게 표시된 이미지는 픽셀을 단위로 저장하는 비트맵(Bitmap)방식으로 저장 장치에 저장된다. 이렇게 저장된 비트맵을 모니터에 나타낼 때는 각 픽셀들을 적색(Red),녹색(Green),청색(Blue)의 값을 적절히 배합시켜 나타내며, 각 픽셀들이 가질 수 있는 칼라의 종류는 각 픽셀에 몇 비트를 할당하느냐에 따라 달라진다. 즉 할당된 비트의 수(Depth)가 클수록 더많은 칼라를 가질수 있고, 할당된 비트의 수가 작을수록 더 적은 칼라를 가지게 되는 것이다. 참고로 1비트 픽셀은 2가지 색상, 즉 하얀색과 검정을 표현하며, 2비트 픽셀은 4가지 색상을 표현한다. 8비트에서는 256가지 색상을 표현하며, 16비트에서 32,768색상, 24비트에서 16,777,216색상을 표현할 수 있다. 즉, 'n비트=2의n승'이라는 공식이 성립된다.

비트수	색상수	비고
1	2	흑백
2	4	팔레트
4	16	팔레트
8	256	팔레트
16	65,536	하이컬라(R:G:B=5:5:5)
24	16,777,216	트루칼라(R:G:B=8:8:8)
32	16,777,216+ 8bit알파채널	트루칼라(24bit)+ 알파채널(8bit)

표를 살펴보면 256가지 이하의 색상에 대해서는 팔레트를 사용하는 것을 알 수 잇다. 여기서 '팔레트'란 사용자가 필요한 색상만을 모아놓은 칼라 네이블로서, 예를 들면 흑백 화상에 대한 팔레트는 검은색부터 흰색까지 순차적으로 변하는 색상들을 포함하고 있다. 32비트 칼라에서는 트루칼라 외에 투명도를 나타낼 수 있는 알파채널을 포함하고 있다. 알파 채널은 RGB채널 이외의 채널로, 256(2의8승) 단계를 가지며 색상 정보를 가지지 않는다. 알파채널이 투명도를 나타낼 때 0은 완전히 불투명한 상태, 256은 완전 투명한 상태를 나타내며 그 중간 값은 반투명을 나타낸다.

B.3.1 파일 사이즈의 계산

이미지 측정단위로 Pixel을 사용하게 되면 실제 파일의 크기를 쉽게 측정할 수 있다. Width가 200, Height가 400pixels이고 24비트인 픽셀 해상도의 파일을 만들 때 실제 파일의 크기는 234KB가 됨을 쉽게 알 수 있다. 계산 방법은 가로*세로*3(8bit는 1, 16bit는 2, 24bit는 3을 곱함)이다. 즉, 200*400*3=240,000byte 이고 이는 약 234KB와 근사 값임을 알 수 있다.

제3절 video4linux를 이용한 실시간 디스플레이

본 절에서는 video4linux를 이용하여 실시간으로 화면을 출력하기 위한 프로그램 개발을 설명한다. OpenGL을 이용한 X Window와 Linux Frame buffer 환경에서 출력이 가능한 두가지의 프로그램을 다룬다.

1. Environment

video4linux를 사용하기 위해서는 우선 하나 이상의 비디오 디바이스와 Linux를 운영체제로 사용하는 컴퓨터가 있어야 함은 당연하다. 필자의 개발 환경 또한 Redhat Linux 9을 설치한 컴퓨터 이며, 비디오 디바이스로 손쉽게 다룰 수 있는 Philips의 Vesta Pro USB Camera를 사용하였다.

USB Camera의 Device Driver는 PWC(http://www.smcc.demon.nl/webcam/)를사용하였으며, 큰 이미지와 높은 프레임 레이트(Frame rate)를 위해서 추가로 PWCX 모듈을 사용하였다.
Redhat Linux 9은 Pentium II 266Mhz의 128MB Ram의 시스템에 개발 도구, 커널 개발, X윈도우 개발 환경등을 추가하여 설치하였으며, 필자가 사용하게 될 OpenGL의 라이브러리와 GLUT라는 유틸리티 툴킷을 추가로 설치하여 OpenGL 프로그래밍 환경을 구성하였다.
또한, 기본적으로 시스템은 런레벨 3을 사용하여 Text 기반의 콘솔 환경으로 부팅이 되도록 하였으며, 32bpp에 800 x 600 해상도의 Linux Vesa Frame buffer를 사용하였다. 필자가 사용한 그래픽 카드는 S3사의 Trio3D로 다행히 Vesa 2.0을 지원하였다.

환경의 구성은 필자와 상이할 수 있지만, 몇가지 특징만 만족한다면 큰 무리 없이 사용이 가능할 것이다.

* RedHat9에서 기본적으로 제공되는 glut 및 glut-devel 패키지는 RedHat9의 기본 X11 라이브러리와 맞지 않는 문제가 있다. 따라서 glut 및 glut-devel을 적합한 버전으로 업그레이드 하여야 한다.

2. video4linux Kit

필자가 개발할 Display 프로그램은 기본적으로 video4linux Kit을 사용한다. video4linux Kit은 video4linux를 이용한 캡쳐 라이브러리 킷으로 더블 버퍼링을 쉽게 사용할 수 있도록 개발 되었다. 따라서 필자는 이 라이브러리를 통해 캡쳐를 하며, 더블 버퍼링 기법을 사용하게 된다. 이와 관련된 자세한 내용은 video4linux Kit Reference를 참조하기 바란다.

3. YUV2RGB

필자가 video4linux Kit을 통하여 캡쳐한 화면은 기본적으로 YUV420P의 포멧으로 구성되어 있다. 이는 필자가 사용한 Philips Vesta Pro가 YUV420P를 잘 지원하기 때문인데, 필자는 각종 포멧에 대한 이해가 부족하기 때문에 자세한 설명은 하기가 힘들다. 하지만, YUV보다 다루기가 쉽고 컴퓨터 상에서 표준으로 사용하는 RGB형태로 데이타를 변환하여 사용하도록 하겠다. 이 변환 과정 중 사용되는 알고리즘은 삼성 MPC-C10 웹 카메라에서 동작하도록 수정된 SIG(Simple Image Grabber)의 알고리즘을 사용한다. 또한 기본적으로 24bpp의 True Color로 이미지를 변환하여 처리하게 된다.

4. Draw Text

video4linux Kit을 이용하여 화면을 캡쳐하고, YUV2RGB를 이용하여 RGB24 형태로 데이타를 변환한 후에 필자는 한가지 이미지 처리를 더 하였다. 이는 이미지의 하단에 푸른색 띠를 입히고, 그 위에 현재 시간과 간단한 멘트를 표시하기 위한 처리인데, 여기에 사용된 drawtext라는 소스와 알고리즘은 Motion이라는 프로그램과 필란드의 한 사이트에서 발견한 소스 파일을 참조하였다.

화면에 텍스트를 표시하기 위한 알고리즘은 사실 매우 간단하다. 가로 6 세로 6의 char 타입의 배열에 0과 1 두가지를 이용한 모노로 폰트를 만들어 저장하고, 이를 이용하여 화면에 표시하게 된다. 각 알파벳 별로 6 x 6의 행렬에 0과1로만 표현된 모노의 디지탈 맵을 구성하고 각각의 알파벳에 해당하는 디지탈 맵을 화면에 하얀색으로 그리게 되는 것이다. 하지만, 글씨가 하얀색 배경에 위치 할 경우 뚜렷하게 보이지 않는 문제점을 해결하기 위해 글씨 주변에 검은색 테두리를 만드는 추가적인 작업이 존재한다. 이 때 주의 할 것은 각 작업은 픽셀 단위로 이루어지고, 대부분의 이미지는 픽셀이 연속되기 때문에 현재 픽셀의 위치가 다음 픽셀이 만든 테두리로 가려질 수 있다는 것이다. 이 때문에 소스에서는 동일한 부분을 테두리를 그리는 반복문과 글씨를 쓰는 반복문으로 나누어 둔 것을 볼 수 있다.

5. XDisplay

X Window를 이용하여 실시간으로 화면을 보여주기 위해서 필자는 우선 윈도우를 만들고 화면을 표시할 수 있는 적절한 도구가 필요했다. 그래서 필자는 OpenGL과 GLUT를 사용하여 이를 해결 하였다.

우선 GLUT를 사용하여 X Window 환경(KDE 환경)에서 새로운 윈도우를 만들 수 있었으며, 미리 캡쳐하여 메모리에 저장된 이미지를 만들어진 윈도우에 그릴 수 있었다. 하지만, 한가지 문제는 OpenGL을 이용하여 이미지를 그릴 경우 윈도우에 거꾸로 뿌려주어야 한다는 것이였다. 아마도 이는 GPU(Graphic Process Unit)이 이미지를 읽을 때 역워드(Inverse Word)방식으로 읽기 때문인 듯 하지만 정확한 원인은 필자도 아직 알지 못한다. 어쨌든 명백한 해결책은 이미지를 좌우/상하로 대칭하여 BGR 형태로 저장하여야 원하는 출력을 얻을 수 있다는 것이다.
또한, OpenGL의 도움으로 이미지 출력시에 더블 버퍼링을 사용할 수 있었다.

6. FBDisplay

X Window 환경에 이어서 Linux Console의 Frame buffer를 사용하여 실시간으로 영상을 보여주기 위해 구현 하였다. 필자의 환경은 S3 Trio 3D 카드를 사용하여 Vesafb를 이용하였다. 하지만 필자가 Vesafb 환경에서 24bpp를 사용하는 방법을 알지 못하는 관계로 32bpp의 Frame buffer 환경을 사용하였다. 따라서 Alpha 채널을 계산하여 32bpp의 이미지로 변환하여야 했다. Frame buffer는 OpenGL과 달리 이미지를 그대로 화면에 뿌려주면 완전한 화면을 볼 수 있어 별도의 화면을 뒤집는 작업은 필요하지 않았다. 그러나 만일 Frame buffer가 MMIO(Memory Mapped Input/Output)을 지원하지 않는다면 부드러운 화면을 보기는 힘들 듯 하다. 이는 Frame buffer를 사용하여 더블 버퍼링을 하는 방법에 대한 필자의 지식이 부족하기 때문이기도 하다.

7. References

[1] Alan Cox, "Video4Linux Kernel API Reference", http://elf.smu.ac.kr/researches/projects/linux-dvr/v4l-kernel-doc/API.html, 1999.

[2] Alex Buell, "Framebuffer HowTo", http://www.tldp.org/HOWTO/Framebuffer-HOWTO.html, 2000.

[3] Ankur Mahajan and Dongha Shin, "Simple Image Grabber", http://pl.smu.ac.kr/lxr/simple-image-grabbler/source/, 2003.

[4] holelee, "Framebuffer 이야기", http://kelp.or.kr/korweblog/search.php?query=frame+buffer+이야기, 2002.

[5] Kim, Boram, "Video4Linux Kit Reference", http://elf.smu.ac.kr/board/?doc=bbs/gnuboard.php&bo_table=documents& page=1&wr_id=20, 2004.

[6] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner, "OpenGL Programming Guide, Third Edition", Addison Wesley, 1999.

[7] Maxwell Sayles, "Video4Linux HowTo", http://pages.cpsc.ucalgary.ca/~sayles/VFL_HowTo/, 2002.

[8] Naoyuki Ichimura, "Image Capture by Video4Linux", http://staff.aist.go.jp/naoyuki.ichimura/research/tips/v4ln_e.htm

[9] GLUT - OpenGL Utility Toolkit, http://www.opengl.org/resources/libraries/glut.html

[10] Motion, http://motion.sourceforge.net

[11] Linux Support for Philips USB webcams, http://www.smcc.demon.nl/webcam

[12] Logitech QuickCam Zoom, http://www.pcuf.fi/~teva/Linux/webcam/webcam_part0.html

[13] OpenGL, http://www.opengl.org

[14] The Mesa 3D Graphics Library, http://www.mesa3d.org

[15] 이국현 역, Vesafb mini HowTo, http://wiki.kldp.org/wiki.php/LinuxdocSgml/Vesafb, 1999.

제4절 X 라이브러리를 사용한 실시간 디스플레이

본 절에서는 X 라이브러리를 처음 접하는 사용자를 위하여 X 윈도우에 영상을 디스플레이 하는 것을 보여주고자 한다. 보다 자세한 X 라이브러리 또는 X 툴킷의 정보는 참고문서를 참고하기 바란다. 그리고 본 문서는 video4linux를 사용해서 영상을 캡쳐한다. 이 문서를 보기 전에 video4linux API의 사용법을 알고 있을것으로 가정한다.

1. X 윈도우 프로그래밍

X윈도우 프로그램이란 창, 색상, 폰트 등의 X윈도우의 자원을 관리하는 X 서버에 서비스를 요청하는 클라이언트를 프로그램 하는 것이라고 할 수 있다. X의 계층적 구조는 X서버 - X 네트워크 프로토콜 - X 라이브러리 - X 툴킷 - 클라이언트 응용 프로그램으로 이루어진다. X 윈도우 프로그래밍은 일반적으로 X 라이브러리와 X툴킷을 혼합하여 사용한다. 그러나 본 문서에서는 X 라이브러리만을 사용할 것이다.

2. 시작하기에 앞서 준비해야 할 것

X 프로그래밍을 하기 위해서 준비해야 할 것은 그리 많지 않다. X 프로그래밍을 하기위해서 최소한 X 윈도우가 작동하고 있어야한다. X 윈도우가 작동하고 있는 환경이라면 X프로그래밍을 할 수 있는 헤더 파일 등이 제대로 설치되어 있는지 확인해보자. /usr/include/X11과 /usr/X11/lib에 화일들이 있는지 확인해보면 된다. X11은 뒤에 버전이 붙어서 X11R6 (Release 6)일수 있다.
그리고 최종 목적이 캡쳐되는 영상을 X 윈도우에서 디스플레이 하는 것이므로 캡쳐를 위한 장비가 있어야 한다.

3. 서버와의 연결

우선 간단한 프로그램을 작성해보자.

```
-------------------------------------------------------------------
 예제 1 - xlib-display01.c
-------------------------------------------------------------------
#include <X11/Xlib.h>

main()
{
  Display *display;
/* Connect display device */
```

```
  display = XOpenDisplay("localhost:0.0");
/* Disconnect display device */
  XCloseDisplay(display);
}
```

위 소스를 컴파일하고 실행시켜보자

```
$ gcc -o xlib-display01 xlib-display01.c -lX11 -I/usr/include/X11 -L/usr/X11R6/lib
$ ./xlib-display01
```

실행시켜보면 아무일도 안 일어날 것이다.
그러나 사실은 X프로그래밍밍을 하기 위한 중요한 작업을 수행 한것이다.

X클라이언트가 서버의 다양한 기능을 사용하기위해 우선 디스플레이에 연결되어야 한다. 이러한 기능은 XOpenDisplay()함수를 통해 제공된다. 이 함수는 디스플레이 이름을 파라메터로 받지만 NULL로 지정하면 환경변수 DISPLAY에 지전된 값을 가져온다. 디스플레이 연결에 성공하면 디스플레이에 대한 정보가 저장된 display형 변수의 포인터를 반환한다.

연결해서 원하는 작업을 마쳤으면 연결을 해제하자. XCloseDisplay()함수를 쓰면 된다. 파라메터로 display변수의 포인터를 주면 된다.

- Display *XOpenDisplay(display)
- XCloseDisplay(Display *display)

거의 대부분의 함수에 display 포인터를 사용하게 될 것이다. 상당히 귀찮을 수도 있지만 서버, 클라이언트 구조의 특성상 다중 디스플레이도 가능하므로 두 개 이상의 디스플레이를 제어하게 된다면 어느 디스플레이에 대한 명령인지를 확실히 해야 하기 때문이다.

4. 윈도우 생성

윈도우를 생성하기 위해 XCreateSimpleWindow()함수를 사용한다. 이 함수는 이름 그대로 간단한 윈도우창을 생성한다.

- Window XCreateSimpleWindow(Display *display, Window parent,
  int x, int y, unsigned int width, unsigned int height,
  unsigned int border_width,
  unsigned long border, unsigned long background)
- XDestroyWindow(Display *display, Window w)

- XDestroySubwindows(Display *display, Window w)

---

display는 디스플레이형 포인터이고 parent는 지금 만드는 창이 속할 부모 창을 지정한다. x,y는 창이 그려질 위치이고 width,height는 창의 크기이다. border_width는 창의 테두리 두께다. border와 background는 테두리색과 배경색이다. 이 함수는 이 속성값으로 윈도우를 생성하고 Window형을 반환한다.

윈도우의 삭제에는 XDestroyWindow()와 XDestroySubwindows()함수 두가지가있다. XDestroyWindow()는 파라메터로 지정한 윈도우를 삭제하고 자식윈도우도 모두 삭제된다. XDestroySubwindows()는 지정한 윈도우의 자식 윈도우를 모두 삭제한다.

여기서 삭제는 단순히 화면에 보이지 않게 하는 것이라 속성까지 모두 삭제되는 것이다.

윈도우를 생성한다고 화면에 보여지는 것은 아니다. 윈도우를 화면에 보이기 위해서는 XMapWindow()함수를 사용한다. 그리고 화면에 보여지고 있는 윈도우를 다시 보이지 않게 하기위해 XUnmapWindow()와 XUnmapSubwindows()를 사용한다.

- XMapWindow(Display *display, Window w)
- XUnmapWindow(Display *display, Window w)
- XUnmapSubwindows(Display *display, Window w)

XUnmapWindow()함수를 사용해 윈도우를 보이지 않게 할때 그 윈도우의 자식윈도우도 모두 보이지 않는 상태가 된다. XUnmapSubwindows()는 자식윈도우를 보이지 않게 한다.

예제 2 - xlib-display02.c

```
#include <X11/Xlib.h>

main()
{
  Display *display;
  Window window, root_window;

  display = XOpenDisplay(NULL);

/* Root window select default window */
  root_window = XDefaultRootWindow(display);
/* Create window - 400*300 size window create, border size 2 and color black */
```

```
  window = XCreateSimpleWindow (display, root_window, 50, 50, 400, 300, 2,
                                BlackPixel(display,0), WhitePixel(display,0) );
/* Display window on screen */
  XMapWindow(display, window);
  XFlush(display);
  getchar();
  XCloseDisplay(display);
}
```

X의 모든 윈도우는 항상 어떤 창에 속해 있게된다. 근본적으로 X윈도우가 뜨자마자 생기는 윈도우는 루트 창이다. 위 예제에서 XDefaultRootWindow()함수는 이 루트창을 알아내는 함수이다.

- Window XDefaultRootWindow(Display *display);

클라이언트 프로그램에서 요청이 발생할 때마다 서버에 요청을 하는게 아니라 버퍼에 저장했다가 버퍼가 채워지면 한번에 보내서 효율성을 높인다. 그러나 버퍼가 가득 차기 전에 요청을 보내야 하는 경우도 있다. 이런 기능을 하는 것이 Flush()함수이다.

- XFlush(Display *display)

윈도우는 자식창을 여러개 가질 수 있다. 그리고 부모 윈도우는 자식윈도우의 좌표의 기준이 된다. 예를 들면 w, w1 두 윈도우가 있고 w는 w1의 부모 윈도우이다. w1을 10,10위치에 생성한다면 w윈도우의 왼쪽 위를 기준으로 10,10위치에 생성한다는 것이다.

5. 색 설정

윈도우를 만들때 사용한 BlackPixel(d, 0), WhitePixel(d, 0)매크로는 검정색,백색을 구하기 위한 매크로였다. 그러나 이제는 우리가 직접 색을 지정하는 함수를 써보자.
서버는 색상을 나타내는 컬러맵이라는 것을 가지고 있다. 컴퓨터는 노랑, 보라 색이 어떤 색인지를 알지 못한다. 단지 각각의 RGB값을 알고 있을 뿐이다. 그래서 노랑, 보라가 어떤 RGB값을 가지고 있는지를 저장하고 있는 표가 있는데 이것이 컬러맵이다.

원하는 색을 지정하기위해 우선 컬러맵의 ID를 알아낸다.

- Colormap XDefaultColormap(Display *display, int screen_no);

---

display로 현재의 서버 screen_no로 현재의 기본 화면(0번화면)을 지정해서 기본 컬러맵 ID를 반환받으면 된다.
이제 이 컬러맵으로부터 우리가 원하는 색의 픽셀값을 알아낼 수 있다.

- Satus XAllocNamedColor(Display *display, Colormap cmap, char *name,
XColor *rcolor, XColor *ecolor)

---

name에는 원하는 색의 이름을 지정하고 색상 정보를 얻는데 XColor형 변수 두개가 들어갔다. rcolor에는 실제 할당된 컬러맵의 픽셀값과 RGB값이 반환되고 ecolor에는 직접 정의한 색의 RGB 값이 반환된다. 색상의 사용은 XColor형 변수의 pixel값을 사용하면 된다.

예제 3 - xlib-display03.c

```
#include <X11/Xlib.h>

int main()
{
  Display *display;
  Window window_main, window1, window2;
  unsigned long black_pixel;
  int window1_position_X, window1_position_Y;
  int window2_position_X, window2_position_Y;
  unsigned int width, height, l;
  Colormap default_colormap;
  XColor color, color1;

  display = XOpenDisplay(NULL);

  black_pixel = BlackPixel(display, 0); /* get black pixel value macro */
  width = 200; height = 100;
  window1_position_X = 10; window1_position_Y = 10;
  window2_position_X = width - 10; window2_position_Y = height - 10;
```

```
/* colormap set default colormap */
   default_colormap = DefaultColormap(display, 0);

/* create main window - 400*200 size window */
   window_main = XCreateSimpleWindow (display, DefaultRootWindow( display ),
                                      100, 100, width*2, height*2, 1,
                                      black_pixel, WhitePixel( display, 0 ));
/* create small window1 - 200*100 size and background fill magenta */
   XAllocNamedColor(display, default_colormap, "magenta", &color, &color1);
   window1 = XCreateSimpleWindow (display, window_main,
                                  window1_position_X, window1_position_Y,
                                  width, height, 1, black_pixel, color.pixel);
/* create small window2 - 200*100 size and background fill blue */
   XAllocNamedColor(display, default_colormap, "blue", &color, &color1);
   window2 = XCreateSimpleWindow (display, window_main,
                                  window2_position_X, window2_position_Y,
                                  width, height, 3, black_pixel, color.pixel);

   XMapWindow(display, window_main);
/* display all window_main's subwindow */
   XMapSubwindows(display, window_main);
   XFlush(display);

   sleep(3);

/* hide window1 */
   XUnmapWindow(display, window1);
   XFlush(display);
   sleep(1);
/* display window1 */
   XMapWindow(display, window1);
   XFlush(display);

   sleep(3);

   XFlush(display);

   XUnmapWindow(display, window_main);
   XUnmapSubwindows(display, window_main);
```

```
  XDestroySubwindows(display, window_main);
  XDestroyWindow(display, window_main);
  XCloseDisplay(display);
  return 0;
}
```

---

6. 그래픽

X윈도우는 GUI체제이므로 그 위의 모든 것은 그리는 것으로 표현된다. 그리기 위해 우선 그리기 위한 공간이 필요하다. 이는 우리가 위에서 만든 Window가 있다. 이것과 Pixmap이라는 자료형을 합쳐 Drawable라는 자료형이 있다. 다음으로 어떻게 그릴 것인지에 대한 정보를 가지고 있는 GC라는 자원을 서버에 요청하고, 그리고 싶은 스타일에 따라 GC의 값을 바꿔 그리면 된다.
윈도우를 만드는 것까지는 위에서 했으니 이제 GC를 만들어보자.

- GC XCreateGC(Display *display, Drawable drw, Unsigned long mask,
  XGCValues *values)
- XChangeGC(Display *display, GC gc, Unsigned long mask, XGCValues values)

Drawable에는 그릴수 있는 타입이 오면 된다. 즉 Window나 pixmap 변수가 오면 된다. mask와 values포인터 타입은 GC의 속성값을 설정하는데 사용한다. 미리 수정할 속성을 XGCValues변수에 지정해주고 각 속성에 해당하는 비트마스크를 OR형식으로 mask에 지정해주면 된다. XGCValues 구조체 형식과 비트 마스크는 Xlib.h와 X.h를 참고하기 바란다.
XCreateGC함수는 GC의 ID를 반환한다. 클라이언트가 서버에 GC의 생성을 요구하면 GC를 서버에 생성하고 ID만을 반환한다. 클라이언트는 그리기 위한 출력함수를 호출할 때 GC의 번호를 지정하여 사용함으로써 출력 함수를 호출할 때마다 속성값을 지정할 필요가 없어 효율적이다.

속성이 설정된 GC가 준비됐으면 이제 그리는 함수를 사용해서 원하는 것을 그리면 된다. 점, 선, 네모, 동그라미 등 각각을 그리기 위한 함수가 있다. 여기서는 점과 네모를 그리는 함수만 소개하고 나머지는 각자에게 맡기겠다.

- XDrawPoint (Display *display, Drawable drw, GC gc, int x, int y)
- XDrawPoints(Display *display, Drawable drw, Gc gc, XPoint *points, int n, int mode)

점을 그리는 함수로 위 두가지가 있다. 전자는 x,y좌표를 받아서 그 자리에 점을 찍고 후자는 XPoint의 구조체로 n개의 점의 좌표를 설정해서 여러 개를 그린다. mode는 CoordModeOrigin(절대좌표)와 CoordModePreivous(상대좌표) 두가지로 절대좌표는 각점을 다 각각의 좌표로 표시하고 상대좌표는 처음의 한점만 원점으로 표현하고 그 다음점부터는 이전의 점을 기준으로 한 상대좌표로 표시한다.

- XDrawRectangle(Display *display, Drawable drw, GC gc,
int x, int y, unsigned int width, unsigned int height)
- XDrawRectangles(Display *display, Drawable drw, Gc gc, XRectangle *rectangles, int n)

점을 그리는 함수와 비슷하다. x,y좌표부터 width, height크기의 네모를 그리는 것과 XRectangle 구조체에 위치와 크기를 설정해서 n개의 네모를 그린다.

다른 색을 사용하고 싶으면 전경색 또는 배경색을 변경한다.

- XSetForeground(Display *display, GC gc, unsigned long fg)
- XSetBackground(Display *display, GC gc, unsigned long bg)

---

fg, bg에 원하는 색의 픽셀값을 넣어주면 된다.
처음에 X의 모든 것은 그리는 것으로 표현된다고 했다.
이것은 텍스트를 출력하는데도 마찬가지이다.
텍스트를 출력하는데는 3가지 과정을 거친다.
우선 폰트를 골라 서버에 그 폰트를 적재하라고 요청한다.

- Font XLoadFont(Display *display, char *font_name)

---

font_name으로는 XLFD식의 폰트 이름 또는 폰트별명을 지정해주면 된다.
이 함수를 호출하고나면 서버에 폰트를 적재하고 폰트형 변수를 반환한다. 다음으로 이것을 GC에 등록한다.

- XSetFont(Dispaly *display, GC gc, Font font)

이제 실제 원하는 텍스트를 출력하면 된다.

- XDrawString(Display *display, Drawable drw, GC gc, int x, int y, char *string, int length)

---

x,y좌표에 그리고자 하는 문자열 string을 주고 스트링의 길이 length를 넣어준다.

예제 4 - xlib-display04.c

```
#include<X11/Xlib.h>

int main()
{
  Display *display;
  Window window;
  Font font;
  GC gc;
  XSetWindowAttributes window_attributes;

  window_attributes.override_redirect = True;
  display = XOpenDisplay(NULL);
  window = XCreateSimpleWindow(display, RootWindow(display, 0), 0, 50, 400, 300, 5,
                                    BlackPixel (display, 0), WhitePixel (display, 0));

  XChangeWindowAttributes(display, window, CWOverrideRedirect, &window_attributes);

  XMapWindow(display, window);

/* Create Graphic Context */
  gc = XCreateGC(display, window, 0L, (XGCValues *)NULL);
/* font select fixed and set */
  font = XLoadFont(display, "fixed");
  XSetFont(display, gc, font);

/* 100*130 position draw string 16 character */
  XDrawString(display, window, gc, 100, 130, "Hello, Linuxers! Never Seen :)", 16);
```

```
  XFlush(display);
  getchar();

  XUnloadFont(display, font);
  XFreeGC(display, gc);
  XDestroyWindow(display, window);
  XCloseDisplay(display);
}
```

---

7. 이미지 출력

영상을 출력하기위해 각 프레임을 이미지로 가져와서 연속적으로 화면에 출력하는 방법을 사용한다.
이미지는 픽셀들의 연속으로 제공된다. 이러한 이미지는 많은 양의 데이터로 구성되기 때문에 XImage 구조체로 X서버에 존재한다.

```
- XImage *XCreateImage(Display *display, Visual *visual,
            unsigned int depth, int format, int offset, char *data,
            unsigned int width, unsigned int height
            int bitmap_pad, int bytes_per_line)
```

처음 보는 내용의 데이터들이 많이 필요하지만 차근차근 보자. visual은 Visual 구조체를 지정한다. 이것은 DefaultVisual()함수를 사용해서 기본값으로 지정해주면 된다.
depth는 이미지의 깊이를 지정한다. 이것도 DefaultDepth()함수를 사용해서 현재 X윈도우에서 사용하는 기본값을 지정해 줄 수 있다.
format은 XYBitmap, XYPixmap, ZPixmap 중의 하나를 지정해준다. offset은 처음에 무시할 픽셀의 수를 지정하고 data에 이미지 데이터에 대한 포인터를 지정하면 된다.
width, height에 이미지의 크기를 지정해준다.
bitmap_pad에 한번에 읽어들이는 라인의 양을 지정하는데 이것은 8, 16, 32 중의 한 값을 지정해야 한다.
bytes_per_line은 한 스캔라인의 첫부분과 그 다음 스캔 라인의 첫부분 사이에 있는 바이트 수를 지정하는 것이다.

예제 5에서 사용한 것을 보면 다음과 같다.

```
img = XCreateImage(dpy, DefaultVisual(dpy, 0), DefaultDepth(dpy, 0),
                        ZPixmap, 0, buffer, 768, 480, 32, 0);
```

이것은 첫 픽셀부터 무시하는 픽셀 없이 사용하고 768*480 크기의 이미지이다. 그리고 한번에 32비트씩의 데이터를 읽어오며 각 라인 사이에 아무 데이터도 없이 계속 픽셀이 이어진다. 이미지 데이터는 buffer포인터에 지정되어 있다. 말로만 설명하니 잘 이해가 안되겠지만 예제를 보고 값을 변경해보면서 각 데이터의 역할을 이해해 주길 바란다.

Ximage구조체를 생성하고 포인터를 받았으면 이제 이것을 화면에 출력한다.

```
---------------------------------------------------------------------
XPutImage(Display *display, Drawable drawable, GC gc, XImage *image,
          int src_x, int src_y, int dst_x, int dst_y,
          unsigned int width, unsigned int height)
---------------------------------------------------------------------
```

src_x, src_y는 이미지의 어느 픽셀부터 출력할 것인지를 나타내고 dst_x, dst_y는 어느 위치에 출력할 것인지를 나타낸다.
width와 height는 출력할 이미지의 크기를 나타낸다.

다음의 예제 5에 video4linux를 사용해 영상을 캡쳐하는 방법은 알 것이라 생각해서 생략했다. 캡쳐해 오는 함수내용은 예제 소스를 참고하기 바란다.
영상을 캡쳐해 올때 주의해야 할 것은 이미지 데이터를 읽을때 8, 16, 32 비트 단위로 읽어 오기 때문에 RGB24와 같은 24비트 데이터로 화면을 캡쳐하게 되면 32비트로 데이터를 수정해 줄 필요가 있다는 것이다.

```
---------------------------------------------------------------------
 예제 5 - xlib-display05.c
---------------------------------------------------------------------
#include <X11/Xlib.h>
#include <unistd.h>
#include <stdio.h>
#include <linux/videodev.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/ioctl.h>

char *v4l_init();
char *v4l_capture();

int fd;
unsigned char *map;
```

```
struct video_mmap v_map;
struct video_mbuf m_buf;

int main( int argc, char * argv[])
{
  Display *display;
  Window window;
  GC gc;
  Visual *visual;
  XImage *display_image;
  unsigned char *buffer;
  int window_width,window_height;

  display = XOpenDisplay(NULL);

/* window size 768*480 */
  window_width = 768; window_height = 480;
  window = XCreateSimpleWindow(display, RootWindow(display,0),
                               50, 50, window_width, window_height, 3,
                               BlackPixel(display,0), WhitePixel(display,0));
  gc = XCreateGC(display, window, 0L, (XGCValues *) NULL);

  XMapWindow(display, window);
  XFlush(display);

  buffer = v4l_init(); /* video capture device initialize */

  while ( 1 )
  {
    buffer = v4l_capture(); /* video capture */

/*
 * image create - depth = default(window depth)
 *                size = 768 * 480
 *                RGB color 32bit (blue, green, red, alpha)
 */
    display_image = XCreateImage(display, DefaultVisual(display, 0),
                                 DefaultDepth(display, 0), ZPixmap, 0,
                                 buffer, 768, 480, 32, 0);
```

```
    display_image->byte_order = LSBFirst;
/* display image */
    XPutImage(display, window, gc, display_image, 0, 0, 0, 0, 768, 480);
  }

  return 0;
}
```

---

8. 참고문서

[1] KLDP wiki, X 윈도우 프로그래밍 기초과정, http://www.kldp.orgwiki.php/LinuxdocSgml/X-Window-Programming-KLDP

[2] UNIX SYSTEM V RELEASE 4 Programmer's Guide: XWIN Graphical Windowing System Xlib-C Language Interface, AT&T, UNIX Software Operation.

[3] 김충석, "X 윈도우 시스템 프로그래밍", 이한출판사, 1994.

제5절 OSS 오디오 프로그래밍

본 절에서는 레코딩 소스를 선택하고 볼륨을 조절하는 믹서(Mixer) 프로그래밍, 레코딩 소스로부터 녹음하여 파일로 저장하고 재생할 수 있는 오디오(Audio) 프로그래밍을 위한 기본 단계를 OSS(Open Sound System) API에 기반하여 설명한다.

1. OSS (Open Sound System)

리눅스 커널에서는 기본적으로 사운드 디바이스 드라이버를 제공한다. 하지만, 리눅스 커널에 기본적으로 포함되어 있는 드라이버는 수가 적어 다양한 사운드 디바이스를 사용할 수 없다. 따라서, 몇가지의 대안이 존재한다.
리눅스의 커널에 기본적으로 포함된 사운드 디바이스 드라이버를 OSS/Free[1]라고 한다. OSS/Free는 4Front Technologies[2]에서 개발한 OSS[3]를 따르는 공개 드라이버로 4Front Technologies의 OSS 개발자가 대부분을 개발하였다. 그러나 다양한 드라이버가 OSS/Free에는 존재하지 않고, 그 외의 많은 드라이버는 4Front Technologies에 의해 상용으로 개발된다. 따라서, 리눅스 진영에서 무료로 사용할 수 있는 드라이버는 그리 많지 않게 된다. 이러한 현실을 극복하기 위해서 SuSE[4] 커뮤니티로 부터 ALSA(Advanced Linux Sound Architecture) Project[5]가 시작되었다. ALSA는 오픈소스 프로젝트로 수 많은 드라이버들이 계속 개발되고 무료로 공개되고 있다.
기본적으로 리눅스에서의 오디오 프로그래밍은 위 세 가지 종류의 디바이스 드라이버에 의존적이다. 하지만, OSS/Free[1]와 OSS[3]는 동일한 OSS API[6]를 사용하고, ALSA[5] 또한 OSS API를 에뮬레이션[7] 하기 때문에 리눅스의 오디오 프로그래밍은 OSS API만으로 충분히 구현 가능하다. 물론, 보다 진보된 기능을 사용하고 싶다면 ALSA Library API[8]를 사용하여야 할 것이다.
OSS 드라이버의 설치 또는 ALSA 드라이버의 설치는 본 문서의 범위를 벗어나기 때문에 필요하다면 Installing Open Sound System[9] 또는 ALSA Sound Card Matrix & INSTALL documentation[10]을 참조하기 바란다.

2. Devices

OSS[3]에는 다양한 디바이스가 존재한다. 기본적으로 유닉스 기반의 시스템은 모든 디바이스를 파일과 동일하게 처리하기 때문에, OSS에 있어서도 예외는 아니다. OSS에 의해서 지원되는 디바이스 파일은 다음과 같다:

- /dev/mixer
  사운드 카드의 믹서에 접근하기 위한 디바이스 파일
- /dev/sequencer
  전자 음악(Electronic Music)을 목적으로 하는 디바이스 파일

- /dev/midi
  Raw 모드로 동작하는 MIDI버스 포트의 인터페이스로 TTY(캐릭터 터미널)과 유사하게 동작하는 디바이스 파일
- /dev/dsp
  디지탈 오디오 어플리케이션(Audio Application)의 주요 디바이스 파일로 기본적으로 8-비트의 부호없는(unsigned) 선형(linear) 인코딩을 사용한다.
- /dev/audio
  /dev/dsp와 동일한 디바이스 파일이지만, 기본 인코딩은 Mu-Law 인코딩이다.
- /dev/dspW
  /dev/dsp와 동일한 디바이스 파일이지만, 기본 인코딩은 16-비트의 부호있는(signed) 리틀-엔디안(little-endian)을 사용한다.
- /dev/sndstat
  다른 디바이스 파일과 달리 사운드 카드의 진단(Diagnostic)을 위한 디바이스 파일로 사람이 읽을 수 있는(Human readable format) 정보를 제공한다. 따라서, 'cat /dev/sndstat'과 같은 명령으로 정보를 볼 수 있다.
- /dev/dmfm
  FM 신디사이저(Synthesizers)를 위한 Raw 인터페이스(Interface) 이다.
- /dev/music
  /dev/sequencer와 매우 유사한 디바이스 파일로 MIDI 디바이스와 신디사이저(Synthesizers)를 동일한 방법으로 다룬다.

위와 같이 OSS에서는 다양한 사운드 디바이스를 지원하지만, 본 문서에서 실제로 사용될 디바이스는 /dev/mixer와 /dev/dsp(또는 /dev/audio나 /dev/dspW)와 같다. /dev/mixer는 각 채널별 음량(Volume)이나 녹음(Recording)의 대상이 되는 디바이스를 설정할 수 있으며, /dev/dsp는 설정된 녹음 디바이스로 부터 데이타를 읽거나 오디오를 출력하기 위해 사용된다.
OSS에서 대부분의 디바이스는 하나 이상 존재하는 것이 가능하다. 따라서 위의 디바이스 파일 뒤에 숫자를 붙여 구분하게 되며, /dev/mixer 또는 /dev/dsp와 같은 파일은 심볼릭 링크(Symbolic Links)로 존재한다. 기본적으로 각 심볼릭 링크는 첫번째 디바이스를 가리킨다(그러나 이를 보장할 수는 없다).
또한 리눅스의 경우 OSS는 모든 디바이스의 메이져 번호(Major Number)를 14로 고정하고, 0에서 8 사이의 번호를 마이너 번호(Minor Number)의 하위 4-비트에 위치시켜 디바이스의 종류를 구분한다. 또한 마이너 번호에서 분류 번호 다음의 4-비트에 디바이스의 번호를 더하여 최종 디바이스의 마이너 번호가 결정된다(예: /dev/dsp1의 마이너 번호는 19(16+ 3:00010011)). 따라서, 실제로 다음과 같은 디바이스 파일이 존재하게 된다:

```
Major     Minor     Name
------------------------------------------------------------------------
14        0         /dev/mixer0
14        1         /dev/sequencer
```

```
14      2       /dev/midi00
14      3       /dev/dsp0
14      4       /dev/audio0
14      5       /dev/dspW0
14      6       /dev/sndstat
14      7       /dev/dmfm0
14      8       /dev/music
14      16      /dev/mixer1
14      19      /dev/dsp1
14      35      /dev/dsp2
```

본 문서에는 /dev/mixer와 /dev/dsp(또는 /dev/audio나 /dev/dspW) 이외의 디바이스에 대한 내용은 다루지 않는다. 자세한 내용이 알고 싶다면 OSS API[6]을 참조하기 바란다.

3. Header Files

리눅스에서 OSS API[6]를 이용한 오디오 프로그래밍을 하기 위해서는 우선 오디오 디바이스와의 통신을 위해 필요한 다양한 함수와 구조체, 상수등을 담고 있는 헤더 파일(Header File)을 include하여야 한다. OSS API에서는 단지 하나의 C언어 헤더 파일 <sys/soundcard.h>만이 요구된다(<linux/soundcard.h>를 사용할 수 있지만, 이는 표준이 아니다). 그러나, OSS API를 사용하기 위한 ioctl, read, 또는 write와 같은 표준 함수나 시스템 콜(System Call)을 사용하기 때문에 다음의 헤더 파일(Header Files)들을 필요로 한다.

```
---------------------------------------------------------------------
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/soundcard.h>
---------------------------------------------------------------------
```

4. Mixer Programming

믹서(Mixer) 프로그래밍은 각종 사운드 채널(Channel)의 음량(Volume)을 설정하고, 녹음(Recording)을 위한 디바이스를 선택하는 방법을 제공한다. 이와 같은 프로그램은 기본적으로 다음과 같은 순서에 따라 실행된다:

1. OSS 믹서 디바이스 열기

2. 사용 가능한 믹서 채널 얻기 (옵션)
3. 사용 가능한 레코딩 디바이스 얻기 (옵션)
4. 레코딩 소스 선택하기
5. 현재의 레코딩 소스 얻기 (옵션)
6. 음량 설정하기
7. 음량 구하기 (옵션)
8. OSS 믹서 디바이스 닫기

이제 위의 각 단계에 대해 자세히 살펴 보겠다.

4.1 Open the OSS Mixer Device

첫번째 단계는 매우 비교적 매우 간단하다. open 함수를 이용하여 믹서(Mixer) 디바이스를 열고 파일 기술자(File Descriptor)를 얻어오면 된다. open 함수를 이용한 코드는 다음과 같다:

```
---------------------------------------------------------------------
int mixer;
char *device = "/dev/mixer";
if ( (mixer = open(device, O_RDWR)) < 0 )
{
  // Could not open device
}
---------------------------------------------------------------------
```

디바이스 이름 device는 기본적으로 /dev/mixer 또는 사용자 입력을 사용하는 것이 좋다. 이는 프로그램의 이식성(Portability)를 높이기 위한 방법으로 프로그래머가 /dev/mixer0과 같은 장치로 값을 고정한다면, 프로그램은 특정 하드웨어에서만 작동하거나 오동작을 할 수 있다. /dev/mixer를 사용할 경우는 심볼릭 링크로 사용자가 변경할 수 있기 때문에 큰 문제가 되지 않는다.
open 호출은 성공하였을 경우 디바이스의 파일 기술자를 mixer로 반환하고, 실패하였을 경우 -1로 설정한다.

4.2 Get Available Mixer Channels (optional)

이번 단계는 생략이 가능하다. 하지만, 모든 사운드 디바이스에서 사용할 수 있는 채널이 같은 것은 아니다. 예를 들어 USB 카메라(Camera)에 내장된 마이크(Mic.)와 같은 사운드 디바이스는 오직 mic만이 가능한 채널이지만, 보편적인 사운드 카드는 보다 많은 채널을 지원한다. 따라서, 범용적인 프로그램의 작성을 위해서는 현재 디바이스의 채널을 알아보아야 할 것이다. 이를 위해 ioctl을 통한 쿼리를 이용하게 된다. 다음의 코드는 사용가능한 믹서 채널을 알아보

기 위해 쿼리를 보낸다:

```
int devmask = 0;
if ( -1 == ioctl(mixer, SOUND_MIXER_READ_DEVMASK, &devmask) )
{
  // Query failed
}
```

ioctl 호출은 성공하였을 경우 0이 아닌 수를 반환하며, 실패하였을 경우 -1을 반환한다.
위에서 사용된 SOUND_MIXER_READ_DEVMASK ioctl 호출은 성공하였을 경우devmask에 사용가능한 채널들의 마스크(Mask)를 담는다. 기본적으로 각 채널은 고유의 상수를 갖는다. 각 상수는 0부터 상수 SOUND_MIXER_NRDEVICES 사이의 정수이다. 따라서, 마스크에 특정 채널의 비트를 확인하기 위해서는 다음의 식을 사용하여야 한다:

```
if ( (1 << SOUND_MIXER_MIC) & devmask )
{
  // SOUND_MIXER_MIC is available.
}
else
{
  // SOUND_MIXER_MIC is not available.
}
```

각 채널에 해당하는 상수는 헤더 파일 <sys/soundcard.h>와 OSS API[6]를 참조하기 바란다.

4.3 Get Available Recording Devices (optional)

이번 단계 또한 생략할 수 있지만, 위의 믹서(Mixer) 채널(Channels)과 동일한 이유로 이식성(Portability)을 위해 사용하는 것이 좋다. 이번 단계에서는 녹음(Recording)을 위해 사용될 수 있는 디바이스를 알아보기 위해 사용된다. 이 또한 ioctl 호출을 이용하여 쿼리를 보내게 되는데 이는 다음의 코드를 사용한다:

```
int recmask = 0;
if ( -1 == ioctl(mixer, SOUND_MIXER_READ_RECMASK, &recmask) )
{
```

```
  // Query failed
}
```

SOUND_MIXER_READ_RECMASK ioctl도 recmask에 마스크(Mask)를 담는다. 따라서 위와 동일한 방법으로 다음과 같이 테스트 할 수 있다.

```
if ( (1 << SOUND_MIXER_MIC) & recmask )
{
  // SOUND_MIXER_MIC is available.
}
else
{
  // SOUND_MIXER_MIC is not available.
}
```

4.4 Select Recording Source

녹음(Recording)에 사용되는 소스(Sources)는 디바이스에 따라 다르지만, 보편적으로 한가지만 사용 가능하다. 녹음을 위한 소스를 선택하기 위해서는 다음의 ioctl 호출을 사용한다:

```
int srcmask = 0;
int src = SOUND_MIXER_MIC;
srcmask = srcmask | (1 << src);
if ( -1 == ioctl(mixer, SOUND_MIXER_WRITE_RECSRC, &srcmask) )
{
  // Set failed
}
```

위의 호출에서 주의할 점은 여러가지 소스를 선택하는 것 또한 가능하기 때문에, srcmask는 마스크(Mask)가 되어야 한다는 것이다. 따라서, srcmask에 선택된 소스의 비트를 추가하기 위해 srcmask = srcmask | (1 <<src) 와 같은 식이 사용된다.
기본적으로 사용자가 각 소스에 해당하는 상수를 인지하기는 어렵다. 따라서, 각 소스에 할당된 고유의 이름이 존재하는데 이는 제공되는 배열 SOUND_DEVICE_LABELS를 통해 다음과 같이 구할 수 있다:

```
int index;
char *src = "mic";
char *names[SOUND_MIXER_NRDEVICES] = SOUND_DEVICE_LABELS;
for (index = 0; index < SOUND_MIXER_NRDEVICES; index++)
{
  if ( 0 == strcmp(names[index], src) )
    srcmask = srcmask | (1 << index);
}
```

4.5 Get Currently Active Recording Sources (optional)

이번 단계에서는 앞서 설정한 녹음(Recording) 소스를 확인하기 위한 방법을 설명한다. 이번 단계 또한 생략이 가능하지만, 정확한 설정이 이루어졌는지 확인하기 위해서 사용하는 것이 좋다. 이번 단계 또한 다음의 ioctl 호출을 사용하여 수행하며, 현재 녹음(Recording) 소스(Source)의 마스크(Mask)를 반환하게 된다:

```
int index;
int recsrc = 0;
if ( -1 == ioctl(mixer, SOUND_MIXER_READ_RECSRC, &recsrc) )
{
  // Query failed
}
for (index = 0; index < SOUND_MIXER_NRDEVICES; index++)
{
  if ( (1 << index) & recsrc )
    // Current recording source
}
```

4.6 Set Volume

이번 단계는 음량을 조절하기 위한 단계이다. 이번 단계에서 음량을 조절하기 위해서는 사운드 카드가 확실하게 지원하는 채널(Channels)을 사용하여야 한다. 채널을 바꾸는 동작 역시 ioctl 호출을 사용하며, MIXER_WRITE() 매크로(Macro)를 사용하여 음량을 조절할 채널을 선택하고 ioctl의 파라미터(Parameter)로 0과 100사이의 음량을 지정하면 된다. 단, 음량은 0과 100사이에서 요구한 볼륨이 정확하게 설정되지 않을 수 있다. 따라서 실제로 설정된 음량이 호출이

완료된 후 파라미터를 통해 반환된다.
만일 채널이 스테레오(Stereo)를 지원할 경우에는 왼쪽과 오른쪽의 음량을 각각 설정할 수 있는데 이 경우에 왼쪽의 음량은 하위 8비트에 오른쪽의 음량은 그 위의 8비트에 저장되게 된다. 또한 모노(Mono)의 경우에는 왼쪽의 음량만을 설정한 채 나머지는 0으로 채우게 된다. 따라서 음량을 설정하기 위해서는 다음의 코드를 이용여야 한다:

```
-------------------------------------------------------------------------
int stereomask = 0;
int volume = 70;
if ( -1 == ioctl(mixer, SOUND_MIXER_READ_STEREODEVS, &stereomask) )
{
  // Query failed
}

if ( (1 << SOUND_MIXER_VOLUME) & stereomask )
  volume = volume | (volume << 8);

if ( -1 == ioctl(mixer, MIXER_WRITE(SOUND_MIXER_VOLUME), &volume) )
{
  // Set failed
}
// Print actual volume
-------------------------------------------------------------------------
```

4.7 Get Volume (optional)

이번 단계는 앞서 설정한 음량이 올바르게 설정되었는지 확인하기 위해서, 사용되는 단계로 생략이 가능하지만, 올바르게 음량이 설정되었는지 또는 현재 설정된 음량을 확인하기 위하여 사용한다. 사용법은 앞의 4.6 Set Volume장에서와 비슷하며, 다음과 같다:

```
-------------------------------------------------------------------------
if ( -1 == ioctl(mixer, MIXER_READ(SOUND_MIXER_VOLUME), &volume) )
{
  // Query failed
}
if ( (1 << SOUND_MIXER_VOLUME) & stereomask )
{
  printf("Current left master volume level is %d₩n", (0xFF & volume) );
  printf("Current right master volume level is %d₩n", (0x00FF & volume) );
```

```
}
else
{
  printf("Current master volume level is %d₩n", (0xFF & volume) );
}
```

4.8 Close the Mixer Device

마지막 단계는 매우 간단하다. 단순히 앞서 open한 믹서(Mixer) 디바이스를 다음의 호출로 닫아주면 된다.

```
close(mixer);
```

---

5. Audio Programming

오디오(Audio) 프로그래밍은 선택된 녹음(Recording) 디바이스로부터 데이타를 읽어와 저장하고, 저장된 데이타를 출력하기 위한 방법을 제공한다. 이와 같은 프로그램은 기본적으로 다음과 같은 순서에 따라 실행된다:

1. OSS 오디오 디바이스 열기
2. 오디오 포멧 (비트수) 설정
3. 채널의 수(모노/스테레오) 설정
4. 샘플링 레이트(Sampling Rate) 설정
5. 녹음(Recording) 및 재생(Playing)
6. 동기화(Synchronize)
7. OSS 오디오 디바이스 닫기

오디오 프로그래밍에 있어서는 위의 단계를 지키는 것이 매우 중요하다. 순서가 뒤바뀔 경우 프로그래머가 원하지 않은 동작을 할 수 있다. 따라서 반드시 위의 순서에 따라 프로그램을 작성하기 바란다. 자세한 이유는 OSS API[6]를 참조하기 바란다.
이제 위의 각 단계에 대해 자세히 살펴 보겠다.

5.1 Open the OSS Audio Device

오디오(Audio) 프로그래밍의 첫번째 단계는 단지 디바이스의 이름이 변경되는 것을 제외하면 믹서(Mixer) 프로그래밍의 첫번째 단계와 동일하다. 또한, 같은 이유로 특정한 디바이스 파일보다는 /dev/dsp와 같은 보편적인 이름을 사용하거나 사용자 입력을 사용할 것을 추천한다. 또

한 녹음(Recording)을 위해 디바이스를 사용하려 한다면 O_RDONLY로, 재생(Playing)을 위해서라면 O_WRONLY로 열어야 한다. 물론 디바이스에 따라 O_RDWR로 디바이스 파일을 오픈할 수 있지만, 이는 Full Duplex를 지원하는 디바이스로 제한되고 별도의 설정을 필요로 한다. 따라서, Full Duplex와 관련된 내용은 본 문서에서 다루지 않으며, 보다 보편적인 Half Duplex를 기준으로 설명하겠다. Full Duplex와 관련된 자세한 내용은 OSS API[6]을 참조하기 바란다.

```
---------------------------------------------------------------------
int audio;
char *device = "/dev/dsp";
if ( (audio = open(device, O_RDONLY)) < 0 ) or
if ( (audio = open(device, O_WRONLY)) < 0 )
{
  // Could not open device
}
---------------------------------------------------------------------
```

5.2 Select Audio Format (Number of Bits)

이번 단계는 오디오 포멧을 설정하는 단계이다. 오디오 포멧은 사운드 데이타의 최소 단위인 샘플(Sample)을 표현하기 위한 방법을 말한다. 오디오 포멧 각각의 포멧에 대응하는 상수가 존재하며, 다음과 같은 오디오 포멧이 OSS에서 제공된다:

Name	Description
AFMT_QUERY	현재의 오디오 포멧을 쿼리한다.
AFMT_MU_LAW	Mu-Law
AFMT_A_LAW	A-Law
AFMT_IMA_ADPCM	16-비트 오디오 시퀀스(Sequence)를 A 4:1로 압축
AFMT_U8	부호없는(unsigned) 8-비트
AFMT_S16_LE	부호있는(signed) 16-비트 리틀-엔디안(Little-Endian)
AFMT_S16_BE	부호있는(signed) 16-비트 빅-엔디안(Big-Endian)
AFMT_S16_NE	부호있는(signed) 16-비트 (시스템 의존적)
AFMT_S8	부호있는(signed) 8-비트
AFMT_S32_LE	부호있는(signed) 32-비트 리틀-엔디안(Little-Endian)
AFMT_S32_BE	부호있는(signed) 32-비트 빅-엔디안(Big-Endian)
AFMT_U16_LE	부호없는(unsigned)16-비트리틀-엔디안(Little-Endian)
AFMT_U16_BE	부호없는(unsigned) 16-비트 빅-엔디안(Big-Endian)
AFMT_MPEG	MPEG MP2/MP3 포멧 (현재 지원되지 않음)

대부분의 사운드 카드는 AFMT_U8 포멧을 지원하며, 요즘 대부분의 사운드 카드에서는 AFMT_S16_LE 포멧을 사용하고 있다. 하지만 부호있는(signed) 16-비트의 경우에는 시스템에 따라 정확한 작동을 보장하지 못할 수 있다. 이에 관한 자세한 정보는 OSS API[6]을 참조하기 바란다. 본 문서에서는 리틀-엔디안 방식의 시스템을 사용한다는 가정아래 가장 보편적인 AFMT_S16_LE를 사용하도록 하겠다. 오디오 포멧을 설정하기 위해서는 다음의 ioctl 호출을 사용한다:

```
------------------------------------------------------------------------
int format = AFMT_S16_LE;
if ( -1 == ioctl(audio, SNDCTL_DSP_SETFMT, &format) )
{
  // Set failed
}
if (AFMT_S16_LE != format)
{
  // Not support
}
------------------------------------------------------------------------
```

오디오 포멧의 설정 또한 믹서(Mixer) 프로그래밍에서의 음량(Volume) 설정과 같이 요청한 오디오 포멧을 지원하지 않는다면, 근접한 오디오 포멧으로 설정하고 파라미터(Parameter)를 통해 그 값을 반환한다. 따라서, 설정 후 적용된 값을 검사하여야 할 필요가 있다.

5.3 Select the Number of Channels (Mono/Stereo)

이번 단계는 우리가 녹음(Recording)하기 위해 선택한 소스의 채널의 수를 설정하는 단계이다. 반드시 이번 단계는 샘플링 레이트(Sampling Rate)의 설정보다 앞서 이루어져야 한다. 자세한 이유는 OSS API[6]를 참조하기 바란다. 채널의 설정 또한 지원하지 않는 채널의 수라면 근사치로 설정하게 되므로 ioctl의 파라미터(Parameter)를 통한 반환된 값을 확인하여야 한다. 채널을 설정하기 위한 ioctl 호출은 다음과 같다:

```
------------------------------------------------------------------------
int channels = 2;
if ( -1 == ioctl(audio, SNDCTL_DSP_CHANNELS, &channels) )
{
  // Set Failed
}
if (2 != channels)
{
```

```
  // Not Support
}
```

5.4 Select Sampling Rate (Speed)

이번 단계는 샘플링 레이트(Sampling Rate)를 결정하는 단계이다. 샘플링 레이트는 소리의 질을 결정하는 중요한 요소 중 하나이다. OSS API[6]는 1Hz에서 2GHz의 대역폭을 지원한다. 기본적으로 지원되는 샘플링 레이트는 8kHz로 전화기 수준의 음질을 뜻하며, CD 수준은 44.1kHz, DVD 수준은 96kHz의 값을 갖는다. 우리는 기본적인 8kHz의 값으로 설정 할 것이며, 디바이스에 따라 지원하는 샘플링 레이트의 값이 서로 다르다. 따라서, 샘플링 레이트의 값 또한 요청한 값에 근사한 값으로 설정될 수 있다. 그러므로, 역시 ioctl 호출 후의 파라미터(Parameter)를 조사하여야 한다. 샘플링 레이트를 설정하기 위한 ioctl 호출은 다음과 같다:

```
int speed = 8000;
if ( -1 == ioctl(audio, SNDCTL_DSP_SPEED, &speed) )
{
  // Set Failed
}
if (8000 != speed)
{
  // Not Support
}
```

5.5 Record or Play

이제 주요 작업인 녹음(Recording)와 재생(Playing)을 하는 단계 이다. 이 단계에서는 우선 오디오(Audio) 데이타를 저장할 버퍼(Buffer)를 생성하여야 하는데, 버퍼의 자료형(Data Type)은 오디오 포멧에 의존적으로 결정된다. 예를 들면, 리틀 엔디안(Little-Endian) 형태의 데이타를 빅 엔디안(Big-Endian) 머신(Machine)에서 사용할 경우 별도의 처리를 위해 데이타 타입이 요구될 수 있다. 또한, 버퍼의 크기는 샘플의 크기의 배수로 설정 되어야 한다.
만약 앞에서 디바이스 파일을 읽기전용으로 열었다면, 프로그래머는 데이타를 읽어 저장하기 위한 녹음(Recording)을 수행하여야 한다. 녹음은 단순히 디바이스 데이타에서 read 호출을 사용하여 정해진 바이트 만큼을 읽어와 파일로 저장하는 작업을 뜻한다. 단, 오디오 디바이스로 부터의 read는 EOF(End of File)가 존재하지 않기 때문에 프로그래머가 정확한 녹음의 끝을 알려야 한다. read 호출을 이용하여 오디오 데이타를 저장하는 방법은 다음과 같다:

```
int index;
int length = 640;
int count = 640;
char *file = "file_name";
FILE *fin;
int flength;
unsigned char buffer[2048];
if ( NULL == (fin = fopen(file, "w")) )
{
  // Could not open device
}
for (index = 0; (index < 1000) && (length > 0) && (flength > 0); index++)
{
  if ( -1 == (length = read(audio, buffer, count)))
  {
    // Read failed
  }
  flength = fwrite(buffer, 1, count, fin);
}
fclose(fin);
```

녹음된 사운드를 재생(Playing)하는 방법은 위와 거의 비슷한 방법으로 write 호출을 사용한다. write 호출의 사용은 다음과 같다.

```
int index;
int length = 640;
int count = 640;
char *file = "file_name";
FILE *fout;
int flength;
unsigned char buffer[2048];
if ( NULL == (fout = fopen(file, "r")) )
{
  // Could not open device
}
flength = fread(buffer, 1, count, fout);
while ( (flength > 0) && (length > 0) )
```

```
{
  if ( -1 == (length = write(audio, buffer, count)))
  {
    // Write failed
  }
  flength = fread(buffer, 1, count, fout);
}
fclose(fout);
```

5.6 Synchronize

이번 단계는 녹음(Recording) 또는 재생(Playing)을 완료한 후 현재 버퍼(Buffer)에 남아있는 내용을 모두 출력 또는 입력하기 위해서 동기화를 시도하는 단계이다. 이 또한 ioctl 호출에 의해서 이루어지는데, 다음의 호출이 버퍼의 내용이 모두 비워질 때까지 프로그램의 동작을 지연(Delay)시키게 된다:

```
if ( -1 == ioctl(audio, SNDCTL_DSP_SYNC, 0) )
{
  // Sync failed
}
```

5.7 Close the OSS Audio Device

마지막으로 믹서(Mixer) 프로그래밍의 경우와 같이 open된 디바이스 파일을 닫기 위해 다음의 코드를 수행하여야 한다:

```
close(audio);
```

---

6. References

[1] 4 Front Technologies, http://www.4front-tech.com

[2] ALSA(Advanced Linux Sound Architecture), http://www.alsa-project.org

[3] ALSA Sound Card Matrix & Install documentation, http://www.alsa-project.org/alsa-doc

[4] Installing Open Sound System, http://www.4front-tech.com/install_gzipped.html
[5] Jaroslav Kysela, Abramo Bagnara, Takashi Iwai, and Frank van de Pol, ALSA Library API on-line documentation, ALSA Project, http://www.alsa-project.org/alsa-doc/alsa-lib/, 2001.
[6] Jeff Tranter, Open Sound System Programmer's Guide, 4 Front Technologies, http://www.opensound.com/pguide/oss.pdf, 2000.
[7] OSS(Open Sound System), http://www.opensound.com
[8] OSS(Open Sound System)/Free, http://www.opensound.com/ossfree/index.html
[9] SuSE, http://www.suse.com
[10] Takashi Iwai, NOTES ON KERNEL OSS-EMULATION, ALSA Project, http://www.alsa-project.org/~iwai/OSS-Emulation.html, 2003.

제6절 DVR 기능 비교

본 절에서는 DVR을 구성하는데 필요한 기능들을 알기 위해 기존에 있는 DVR 프로그램들의 기능을 확인하고 비교하여 정리한 문서이다. 여기에서 비교한 DVR 프로그램은 윈도우의 Fusion PVR, MyHTPC와 리눅스의 MythTV, 그리고 Tivo Basic, ReplayTV4500이다.

<표 2.1> DVR 기능 비교

레코딩 기능	Fusion PVR	MythTV	MyHTPC	Tivo Basic	ReplayTV 4500
비디오 화질 조정	O		X	O	O
다채널 동시 녹화	X	O	X	X	X
뷰와 동일한 화질 녹화	O		O	X	O
레코딩 포맷	MPEG				
스케쥴 기능	Fusion PVR	MythTV	MyHTPC	Tivo Basic	ReplayTV 4500
시리즈 레코딩 (드라마 등)	O		X	X	O
레코딩 목록 조회	O	O	O	O	X
삭제 목록 조회	X		X	O	X
예약 시간 충돌 방지 (새로운 예약시)	O	O	O	O	O
예약 시간 충돌 방지 (예약 변경시)	O	O	X	X	X
예약 우선 순위 설정	X	O	O	X	X
날짜/시간/채널 (VCR방식) 예약	O	O	X	O	O
특정 요일로 레코딩 제한	O	O	X	X	O
EPG	O	O	O	O	O
EPG 기반 레코딩 예약	O	O	O	O	O
EPG 소스	Unknown Internet	XMLTV	XMLTV	Tivo	ReplayTV
실시간 뷰 기능	Fusion PVR	MythTV	MyHTPC	Tivo Basic	ReplayTV 4500
실시간 뷰 버퍼 (Time Shifting)	X	O	X	O	O
버퍼를 저장 가능	X		X	O	X

플레이 기능	Fusion PVR	MythTV	MyHTPC	Tivo Basic	ReplayTV 4500
자동 광고 스킵	X	O	X	X	O
수동 광고 스킵(단일 버튼)	X		X	X	X
앞으로 또는 뒤로감기 속도 조절	X		X	O	O
슬로우 모션	X		X	O	O
프로그램 정보 보기	X	O	X		
Picture in Picture	X	O	X	X	X
플레이 리스트	X	O	O	O	O
검색 및 EPG 자료 기능	Fusion PVR	MythTV	MyHTPC	Tivo Basic	ReplayTV 4500
키워드 기반 검색	O		O	X	O
카테고리별 검색	O		X	X	X
다중 키워드 검색	O		X	X	X
카테고리별 목록 보기	O	O	X	O	O
EPG 필터링	X		O	X	X
네트워킹 기능	Fusion PVR	MythTV	MyHTPC	Tivo Basic	ReplayTV 4500
웹을 통한 설정	X	O	X	X	O
웹을 통한 뷰	X	O	X	X	X
웹을 통한 플레이	X		X	X	X
분산 레코딩 및 플레이 지원	X	O	X	X	X
기타 기능	Fusion PVR	MythTV	MyHTPC	Tivo Basic	ReplayTV 4500
비디오 편집	X	O	X		
디스크 사용량 보기	X	O	X		
HDTV 지원	X	O	X	X	X
레코딩된 자료를 카테고리 및 폴더로 분류 가능	X	O	X	O	O
디지틀 사진 슬라이드 쇼	X	O	O	O	O
MP3 플레이	X	O	O	O	X
스킨 기능	X	O	O		
날씨 보기	X	O	O		
DVD 플레이	X	O	X		X
운영체제	Windows	Linux	Windows	Linux	Unknown

제3장 요구 분석 단계

본 장에서는 "리눅스 디지틀 비디오 레코더"의 요구 사항을 기술한다. 본 과제의 소프트웨어는 "리눅스 디지틀 비디오 레코더"이며, 이 중 TV 관련 부분은 DVR TV라 하고, Motion 관련 부분은 DVR Motion이라 한다. 1절에서는 "리눅스 디지틀 비디오 레코더" 소프트웨어 사용에 필요한 환경에 대해 기술한다. 2절에서는 DVR TV의 요구사항을 기술하고, 3절에서는 DVR Motion의 요구사항을 기술한다. 마지막 4절에서는 사용자 인터페이스 및 웹 인터페이스에 관해 기술한다.

제1절 사용 환경

본 절에서는 "리눅스 디지틀 비디오 레코더"의 사용에 필요한 운영 환경을 기술한다.

1. 하드웨어(Hardware)

1.1 CPU

Linux DVR의 동작은 Intel의 x86 플랫폼을 기반으로 한다. Linux DVR은 소프트웨어 비디오 인코딩/디코딩의 사용을 기본 전제로 한다. 따라서 CPU의 성능에 따라 프로그램의 수행 능력이 크게 영향을 받는다.
MPEG4 포멧을 사용하여 한 가지 소스(TV 저장)를 인코딩 할 경우 실험적으로 Pentium III 1Ghz의 CPU로 충분히 수행이 가능하다.또한, LinuxDVR은 동시에 두가지 이상의 소스를 인코딩 하는 것이 가능하기 때문에 보다 많은 디바이스를 이용한다면, 필요한 사양은 비례하여 증가한다.

1.2 메모리(Memory)

CPU의 경우와 같이 단일 소스를 인코딩 하기 위해서는 256MB의 메인 메모리면 충분하다. 하지만, USB 카메라를 이용한 인코딩 및 저장과 같이 많은 디바이스를 사용 할 경우 더 높은 자원이 요구된다.

1.3 하드 디스크(Stroage)

하드 디스크는 사용자가 저장하기를 원하는 자료의 양에 의존적이다. 기본적으로 화질의 열화 없는 실시간 버퍼링 재생을 위해 약 2GB 정도의 하드 디스크가 요구되며, MPEG4 포멧으로 약 1시간의 동영상을 저장하게 되면 약 700MB의 디스크가 추가로 요구된다. 따라서, 설치 및

프로그램의 동작에 추천되는 디스크 공간의 크기는 레코딩과 USB 카메라를 고려하여 약 5GB로 한다.

1.4 비디오 카드(Video card)

Linux DVR에서 사용하게 되는 사용자 인터페이스는 XFree86[1]에 기반하여 개발하게 된다. 따라서, 사용자의 비디오 카드는 XFree86에서 지원할 수 있는 그래픽 카드로 한정된다. 하지만 XFree86이 프레임버퍼를 지원하기 때문에 Vesa 2.0 이상과 호환되는 비디오 카드라면 모두 사용이 가능할 것이다.
Linux DVR에서 사용하게 될 XFree86의 기본 버전은 4.3.0이며, 해당 버전에서 지원되는 비디오 카드 드라이버의 벤더는 3DFX, ATI, Cirrus Logic, Intel, Matrox, NVIDIA, S3 등으로 자세한 비디오 카드 목록은 XFree86의 홈페이지 http://www.xfree86.org/current/Status.html을 참조하기 바란다.
개발 및 테스트에 사용되는 비디오 카드는 기본적으로 Intel Graphic Extreme 2 865G 이다.

1.5 사운드 카드(Sound card)

Linux DVR의 사운드는 Linux의 OSS(Open Sound System)[2]를 기반으로 한다. 따라서, OSS 방식으로 작동되는 사운드 카드는 모두 사용이 가능하다.
Linux DVR에서 사용되는 사운드 디바이스는 TV의 소리를 출력하거나 레코딩된 소리를 출력하기 위한 목적으로만 사용되고, TV의 녹음을 위해서는 TV카드를 이용하게 된다. 따라서, 사운드 카드는 OSS방식의 믹서 디바이스(/dev/mixer)와 코덱 디바이스(/dev/dsp)만을 제공하면 된다. OSS 버전 OSS/Free 3.8을 기준으로 지원이 가능한 디바이스는 대표적으로 SoundBlaster 호환과 Adlib사의 디바이스 등이 존재하며 자세한 사운드 카드 목록은 http://www.linux.org.uk/OSS를 참조하기 바란다.
개발 및 테스트에 사용되는 사운드 카드는 기본적으로 ShuttleX SB61G2의 메인보드에 온보드된 RealTek ALC650 칩과 AC97 Codec 이다.

1.6 TV 카드(TV card)

Linux DVR에서 지원하게 되는 TV 카드는 Linux의 bttv 드라이버[3]를 사용할 수 있는 BT878 기반(BT878, Fusion 878a 등)의 TV 카드이다. 튜너에 사용된 칩 또한 BTTV의 카드 리스트에 나열되어 있거나 호환되어야 한다. 또한 TV 카드를 통하여 사운드를 녹음하기 위해서는 BTAUDIO 드라이버를 지원하여야 하며, 이는 BTTV를 지원하는 위의 카드를 사용한다면 기본적으로 지원된다.
국내 실정에 맞는 자세한 카드의 목록은 구하기 쉽지 않지만, 위의 조건에 만족한다면 올바르게 작동할 수 있을 것이다. 개발 및 테스트에 사용되는 TV 카드는 기본적으로 Fusion 878a 칩셋을 사용한다.

1.7 USB 카메라(USB Camera)

Linux DVR에서 사용되는 USB 카메라 드라이버는 옴니비젼 511[4]과 필립스 웹 카메라[5] 드라이버이다. 따라서, 옴니 비전 또는 필립스의 칩셋을 사용하는 카메라일 경우 사용이 가능하다.
개발 및 테스트에 사용되는 USB 카메라는 OV511을 지원하는 알파캠, 삼성 C-90 및 PWC를 지원하는 Logitech QuickCam Pro 3000, 4000, Phillips Vesta Pro 등이다.

2. 리눅스 커널(Linux kernel)

Linux DVR은 기본적으로 리눅스 커널 2.4[5]를 기반으로 하여 제작되지만, 커널 2.6에서도 정상 작동할 수 있도록 개발 된다. 즉, 커널 2.6에서만 사용이
가능한 기능은 제외하고 제작된다. 개발 및 테스트에 사용되는 리눅스 버전은 2.4.20 이상이다.

3. 디바이스 드라이버(Device drivers)

3.1 BTTV

TV 카드의 드라이버는 BT848과 호환되는 BT878 카드의 사용을 위해 BTTV드라이버[3]를 사용하며, 사운드의 녹음을 위해 BTAUDIO 장치를 사용한다. 현재의 리눅스 커널 2.4에서 기본적으로 포함되어 있는 BTTV의 버전은 0.7.108 이며, 이 버전 이상의 드라이버에서 정상 작동하도록 개발 할 계획이다.

3.2 OSS/ALSA

사운드 부분은 앞에서 언급했듯이 커널 버전 2.4에 기본 포함된 OSS/Free[7] 3.8을 이용하여 개발한다. 그러나 OSS/Free만을 이용하여 사용자의 사운드 카드를 정상 설치할 수 없거나 ALSA[8]가 정식 포함된 커널 버전 2.6을 사용하는 경우에는 ALSA 드라이버의 OSS 에뮬레이션 기능을 이용하여, OSS 디바이스와 호환될 수 있는 환경을 만들어 사용한다.

3.3 OV511/PWC

USB 카메라를 지원하는 옴니비젼과 필립스의 드라이버는 기본적으로 리눅스 커널 2.4에 포함되어 있다. 사용되는 버전은 OV511 1.63[4]과 PWC는 8.11[5]을 사용한다. 커널 버전 2.6의 경우에는 더 상위버전의 드라이버를 사용하게 될 것이다.

3.4 video4linux

각종 비디오 장치를 위한 video4linux[9] 드라이버는 기본적으로 커널 2.4의video4linux 1.x를

사용한다. 커널 버전 2.6에서는 video4linux 2를 지원하지만 video4linux 1도 지원하므로 하위 호환성을 위해 video4linux 1을 사용한다.

4. 라이브러리 및 응용프로그램(Libraries and applications)

4.1 ffmpeg

각종 소프트웨어 비디오 인코딩과 디코딩을 위해서 사용하는 그래픽 라이브러리는 ffmpeg[10]이며, 개발 및 테스트에 사용될 버전은 0.4.8 이상의 버전이다. 사용될 코덱은 MPEG4 또는 다양한 고려에 따른 또 다른 코덱이 될 수 있다.

4.2 XFree86

Linux DVR의 사용자 인터페이스의 개발에 사용하는 그래픽 라이브러리는XFree86의 리눅스 기본 X 라이브러리를 사용한다. 경우에 따라 효율성을 위해 추가적인 그래픽 툴킷의 사용도 고려한다. 개발에 사용하는 XFree86 버전은 4.3.0 이상이다.

제2절 DVR TV 요구사항

1. 실시간 뷰(View)

실시간 뷰는 일반 TV가 가지고 있는 기능을 말한다. 채널 및 볼륨 조정, 채널 미세 조정, 채널 자동 스캔 기능들이 여기에 포함된다. 여기에 컴퓨터의 기록 능력을 활용하여 방송되는 내용을 일시정지시켰다 다시 보거나 조금 이전 또는 이후의 내용으로 돌아갈 수 있는 기능(Time Shifting)이 포함된다. 현재 버퍼링된 내용은 기록이 가능하며, 버퍼링을 통한 재생시 화질의 손실은 최소화한다. 추가로 간단한 화면의 정보를 표시할 수도 있다.

2. 기록(Record)

TV와 관련하여 제공될 수 있는 기록 기능은 현재 TV를 시청하면서 그 내용을 기록하는 것과 시간 예약을 통해 TV 프로그램을 기록하는 기능 등이 있다. 우선 TV를 시청하면서 사용자가 "기록"버튼을 누르면 기본 기록폴더에 기록이 시작되고 "정지"버튼을 누르면 기록이 완료되고 파일명을 정할 수 있게 해준다. 둘째로 시간 예약을 통해서 TV 프로그램을 기록하는 기능은 사용자가 원하는 채널과 시간대를 설정하면 기록이 이루어지는 기능이다.

예약 기록 방식에는 시간으로 예약하는 방법과 TV 가이드를 이용하는 방법이 있다. 하지만 TV 가이드를 이용하는 방식은 이번 버전에서는 구현하지 않고 추후 버젼에서 추가하기로 한다.

3. 재생(Play)

기존에 기록했던 파일을 보고자 할 때 사용하는 것으로 검색기능을 이용하여 파일을 선택하고 선택된 파일을 재생한다.
화면은 full screen으로 한다.

4. 관리기능

4.1 파일관리

파일 관리에는 파일 찾기, 파일 지우기, 파일 이름 변경하기, 파일 옮기기 등이 있다. 나머지 파일 관리는 추후 추가할 예정이다.

4.2 환경설정

- 기본 기록폴더 설정

- 기록시 저장 장치의 용량이 포화될 경우 이전 파일을 삭제하거나 기록을 멈춘다.
- 4절의 메뉴리스트의 TV환경설정 참조(1.1.3)

제3절 DVR Motion 요구사항

1. 실시간 뷰(View)

카메라로 현재 촬영되는 영상을 실시간으로 화면에 표시하며, 모션 검출 기능에 의해서 검출된 부분을 표시할 수 있다.

2. 기록(Record)

Camera와 관련하여 제공될 수 있는 기록 기능은 세 가지가 있다. 무조건 기록하기, 무조건 기록하지 않기, 그리고 예약 기록하기가 있다. 무조건 기록하기가 설정(ON)되면 Camera 영상은 기본 기록폴더에 기록이 된다. 무조건 기록하지 않기가 설정(무조건 기록하기가 OFF)되면 Camera 영상은 기록되지 않는다. 예약 기록하기 기능은 사용자가 시간대를 설정하면 기본 기록폴더에 그 내용이 기록된다.

3. 재생(Play)

기존에 기록된 파일을 보고자 할 때 사용하는 것으로 검색기능을 이용하여 파일을 선택하고 선택된 파일을 재생한다.
화면은 full screen으로 한다.

4. 모션 검출(Motion detection)

Camera 입력에서 움직이는 물체의 검출 기능이 있다.

4.1 모션 검출에 관한 기능

본 프로젝트에서 모션 검출에 관한 기능은 모션 검출 ON 기능과 모션 검출 OFF 기능이 있다.

- 모션 검출 ON 기능

모션 검출 ON 기능은 실시간으로 카메라의 화면을 보여주다가 카메라에서 움직임이 검출되면, 화면에 모션 검출된 부분을 표시해준다.

화면에 모션 검출된 부분을 표시하는 방법으로는 모션 검출된 부분만 검은색으로 표시하거나 모션 검출된 부분만 상자로 표시하는 방법이 있다. 모션 검출된 부분의 표시 방법은 추후 결정한다.

- 모션 검출 OFF 기능
모션 검출 OFF 기능은 카메라에서 움직임이 나타나도 아무런 표시 없이 단순히 실시간으로 화면을 보여준다.

4.2 모션 검출 방법

카네기 멜론 대학의 로봇 공학 연구소(The Robotics Institute at Carnegie Mellon University)에서는 VSAM(Video Surveillance and Monitoring)[11]을 개발하였다. VSAM에서 움직이는 물체의 검출을 위한 하이브리드 알고리즘(Hybrid Algorithm)을 연구한 부분이 있다. 이 알고리즘은 움직이는 물체를 검출해내는 것으로써 현재 이미지와 한 단계 이전 이미지, 현재 이미지와 두 단계 이전 이미지 그리고 현재 이미지와 background 이미지의 각 픽셀 차이를 보고 임계값보다 크면 픽셀이 움직인 것으로 보고 그렇지 않으면 픽셀이 움직이지 않은 것으로 본다.

본 프로젝트의 모션 검출은 위의 하이브리드 알고리즘을 기반으로 구현한다.

5. 관리기능

5.1 파일관리

파일 관리에는 파일 찾기, 파일 지우기, 파일 이름 변경하기, 파일 옮기기 등이 있다. 나머지 파일 관리는 추후 추가할 예정이다.

5.2 환경설정

- 기본 기록폴더 설정
- 기록시 저장 장치의 용량이 포화될 경우 이전 파일을 삭제하거나 기록을 멈춘다.
- 모션 검출된 부분을 기록하기 원하면 모션 검출된 화면만 디스크에 기록한다.
- 4절의 메뉴리스트의 Camera 환경설정 참조(1.2.3)

제4절 사용자 인터페이스 및 웹 인터페이스

UI는 X윈도우에서 사용할 수 있는 GUI(Graphic User Interface)를 사용한다.

1. 메뉴의 구성

다음은 메뉴의 구성을 나타낸다. 아래 기술한 메뉴의 구성은 추후 변경될 수도 있다.

1.1 TV

1.1.1 뷰(View)

- 실시간 뷰 및 재생
  - 재생, 멈춤, 타임 쉬프팅(time shifting)
  - 채널변경(TV)
  - 볼륨
  - 저장(TV)
  - 화면조정
    - 크기, 밝기, 컨트라스트
- 파일관리
  - 재생
  - 탐색
  - 삭제
  - 이동

1.1.2 기록(record)

- 예약
  - 시간
- 예약리스트
  - 탐색
  - 삭제
  - 수정

1.1.3 환경설정(Setup)

- view
  - 입력포맷

    - NTSC, PAL
  - 화질
  - 채널리스트
  - 버퍼
- 재생
  - 앞/뒤 점프 양
- 저장
  - 저장공간
    - 디렉토리, 최대기록용량
  - 포맷
    - 퀄리티
    - 크기, 프레임레이트

1.2 Camera

1.2.1 뷰(View)
- 재생
  - 재생, 멈춤, 점프(replay시에 사용)
  - 볼륨
  - 저장
  - 화면조정
    - 크기, 밝기, 컨트라스트
  - 모션 on/off
- 파일관리
  - 재생
  - 탐색
  - 삭제
  - 이동

1.2.2 기록(record)
- 예약
  - 시간
- 예약리스트
  - 탐색
  - 삭제
  - 수정

1.2.3 환경설정(Setup)
- Camera

  - 입력포맷
    - S-Video, Composite
  - 화질
  - 버퍼
- 기록
  - 기록공간
    - 디렉토리, 최대기록용량
- 포맷
- 퀄리티
  - 크기, 프레임레이트

2. Web Interface

- 실시간 보기
  - 채널/입력 변경
  - 모션 on/off
- 저장된 영상 재생
  - 파일 선택
  - 모션 on/off

* 3장 참고문서

[1] Advanced Linux Sound Architecture, http://www.alsa-project.org

[2] BTTV, http://bytesex.org/bttv

[3] ffmpeg, http://ffmpeg.sourceforge.net

[4] Linux OVCam Drivers, http://alpha.dyndns.org/ov511

[5] Linux support for Philips USB webcams, http://www.smcc.demon.nl/webcam

[6] Linux Kernel, http://www.kernel.org

[7] Open Sound System, http://www.opensound.com

[8] Open Sound System/Free, http://www.opensound.com/ossfree/index.html

[9] Robert T. Collins, Alan J. Lipton, Takeo Kanade, Hironobu Fujiyoshi, David Duggins, Yanghai Tsin, David Tolliver, Nobuyoshi Enomoto, Osamu Hasegawa, Peter Burt and Lambert Wixson, "A System for Video Surveillance and Monitoring: VSAM Final Report," Technical report CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, May, 2000.

[10] Video4Linux resources, http://www.exploits.org/v4l

[11] XFree86, http://www.xfree86.org

제4장 설계 단계

본 장에서는 리눅스 디지틀 비디오 레코더 TV 부분과 Motion 부분 설계에 관해 기술한다. 리눅스 디지틀 비디오 레코더 TV를 DVR TV라 하고 리눅스 디지틀 비디오 레코더 Motion은 DVR Motion이라 한다. 1절과 2절에서는 DVR TV와 DVR Motion의 각각 구조와 자료 흐름 및 자료구조와 알고리즘에 관해 기술한다. 마지막 3절에서는 설계 단계 기술 문서를 소개한다. 설계 단계 기술 문서로는 X 라이브러리와 LessTif을 사용한 디스플레이 방법, ffmpeg을 사용한 mpeg 인코딩 방법, ffmpeg을 사용한 mpeg 디코딩 방법, 그리고 OSS와 ffmpeg을 사용한 mp2 오디오 레코딩 방법이 있다.

제1절 DVR TV 설계

본 절에서는 DVR TV의 구조 및 기능과 프로세스별 자료 흐름, 프로세스별 및 세부 함수 알고리즘에 관해 기술한다.

1. DVR TV 구조 및 기능

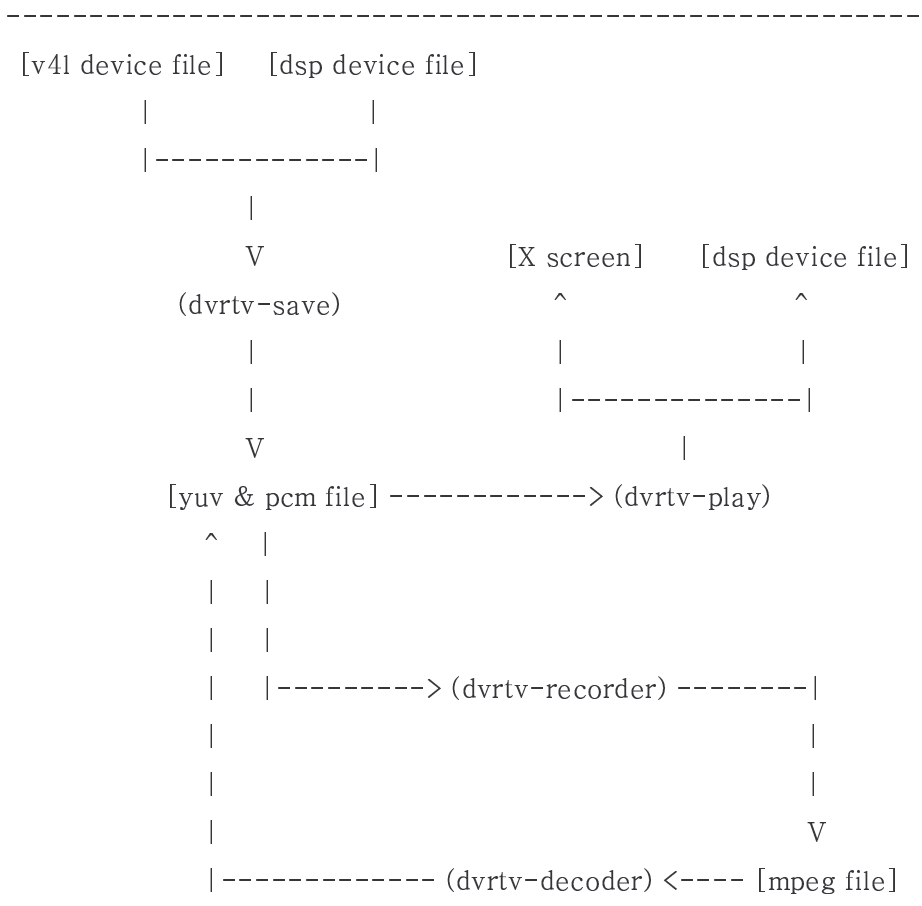
여기서는 Linux DVR의 TV 부분의 전체적인 구조를 그림으로 표현하고, TV의 기능을 기술한다.

1.1 DVR TV 구조

dvrtv-save에서 video4linux에서 yuv이미지 데이터, OSS에서 pcm사운드 데이터를 가져와 저장한다. 저장한 yuv & pcm file은 dvrtv-play에서 X screen에 yuv이미지 데이터를 뿌려주고, OSS 드라이버에 pcm사운드 데이터를 뿌려준다.
dvrtv-recoder는 dvrtv-save가 저장한 yuv & pcm file에 접근하여 yuv 이미지 데이터를 mpeg1video코덱을 사용하고, pcm사운드 데이터는 mp2코덱을 사용해서 mpeg을 만든다.
dvrtv-decoder는 만들어진 mpeg파일을 디코딩 수행하여 yuv & pcm file을 만든다.

데이터, 프로세스, 자료 흐름, 프로세스 실행은 다음과 같이 표현한다.

데이터: [데이터 명]
프로세스: (프로세스 명)
자료 흐름: 화살표(-->)
프로세스 실행: 선(---)
프로세스간 공유 방법: <공유 방법 명>



1.2 DVR TV 기능

본 프로젝트에서 TV 부분은 다음의 기능을 가지고 있다.

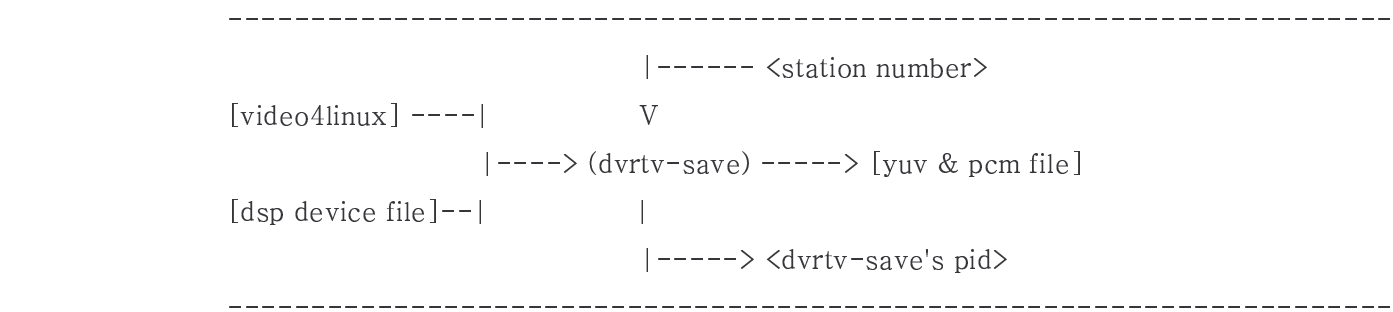
- 실시간 뷰
 . 볼륨 조절
 . 채널 조절
 . 일시정지
 . 슬로우모션
 . 타임 쉬프팅
 . 간단한 화면 정보 표시
 . 화면 확대 축소
- 기록
 . 실시간 녹화
 . 예약 녹화
- 재생

2. 프로세스별 자료흐름

여기서는 DVR TV의 핵심 프로세스들의 자료 흐름에 대해서 설명한다. 프로그램을 수행시킬 때, 생성되는 프로세스는 모두 5가지로 dvrtv-save, dvrtv-play, dvrtv-recorder, dvrtv-decoder, dvrtv-reservation이 있다.

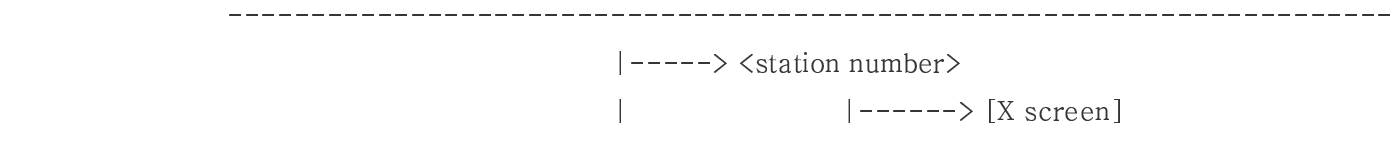
2.1 dvrtv-save

dvrtv-save 프로세스는 video4linux로 부터 yuv이미지 데이터와 dsp device file로 부터 pcm 사운드 데이터를 읽어와 버퍼 파일에 저장한다. 여기서 버퍼 파일은 원하는 갯수만큼 만들수 있으며 파일포인터는 1개를 사용하여 현재 사용되고 있는 파일만 열려 있다. 또한, 한개 파일에 들어가는 이미지 수 또한 정해 줄 수 있다.
본 프로세스는 dvrtv-play와 특정 정보를 공유하기 위해 공유메모리(shared memory)를 사용한다. 공유메모리가 쓰이는 가장 큰 이유는 TV의 채널변경 기능때문이다. dvrtv-save는 생성될 때 자신의 프로세스 ID(pid)를 공유메모리에 저장하고, dvrtv-play로 부터 채널변경 시그널을 받을 경우 공유메모리로 부터 방송국 채널번호(station number)를 읽어와 채널을 변경한다.



2.2 dvrtv-play

dvrtv-play 프로세스는 Storage Device에 저장되어 있는 yuv이미지 데이터를 XVideo로 넘겨주게 된다. 그러면 X는 넘겨받은 데이터를 X화면에 뿌려주게 된다. dvrtv-play 프로세스는 dvrtv-save와는 달리 현재 찍어야할 이미지를 놓칠 경우 무시하고 넘어간다.
본 프로세스 또한 dvrtv-save와 마찬가지로 채널변경 때문에 공유메모리를 사용한다. 다른 점은 자료의 이동방향이 dvrtv-save와는 반대라는 것이다.
dvrtv-play는 사용자로 부터 채널을 변경하라는 입력을 받을 경우, 공유 메모리에 먼저 변경하고자 하는 방송국 번호를 쓰고, dvrtv-save에서 또 다른 공유메모리에 저장한 자신의 pid를 읽어와 dvrtv-save로 시그널을 보낸다.



```
[yuv & pcm file] -----> (dvrtv-play) ------|
                              ^                |------> [dsp device file]
                              |------ <dvrtv-save's pid>
```

2.3 dvrtv-recorder

dvrtv-recorder 프로세스는 yuv & pcm file에 저장되어 있는 yuv이미지 데이터와 pcm사운드 데이터를 읽어서 mpeg 파일을 만든다.

```
[yuv & pcm file] ------> (dvrtv-recoder) ------> [mpeg file]
```

2.4 dvrtv-decoder

dvrtv-decoder는 dvrtv-paly에서 실행시키는 쓰레드다. mpeg 파일을 읽어와 yuv & pcm file을 만든다.

```
[mpeg file] --------> (dvrtv-decoder) -------> [yuv & pcm file]
```

2.5 dvrtv-reservation

dvrtv-reservation 프로세스는 문자열로 입력받은 예약 정보를 구조체에 가지고 있으면서 시간을 확인해서 예약 시간과 일치하면 DVR TV를 실행, 종료 한다.

```
          Time Check
 |--------------------|
 |                          |   system call
 |->   dvrtv-reservation --|----------------- DVR TV
```

---

3. 프로세스별 알고리즘

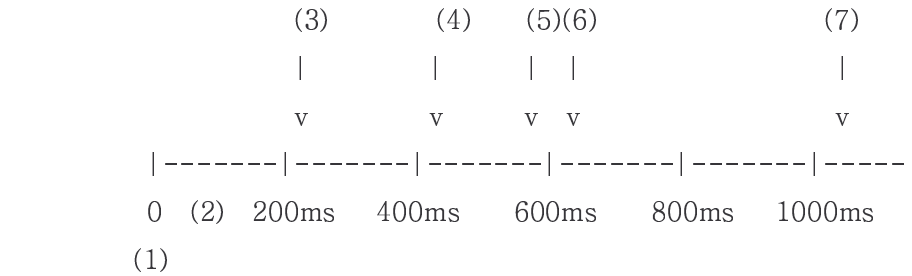
3.1 전체 자료구조

3.1.1 상수

- WIDTH: 캡쳐된 이미지의 가로 길이(320)
- HEIGHT: 캡쳐된 이미지의 세로 길이(240)
- FRAME_PER_SEC: 초당 캡쳐하고자 할 이미지의 개수(25)
- VIDEO_SEC_DATA_SIZE: 초당 비디오 데이터 사이즈
- SAMPLE_RATE: 오디오 샘플 레이트(44100)
- CHANNELS: 오디오 채널 수(1:mono, 2:stereo)
- OSS_FORMAT: 샘플 포맷(AFMT_S16_LE)
- AUDIO_SEC_DATA_SIZE: 오디오의 초당 데이터 사이즈
- SEC_DATA_SIZE: 초당 데이터 사이즈
- JUMP_TIME: 쉬프팅 한번당 점프하는 시간(단위: 초)
- JUMP: 쉬프팅 한번당 점프하는 데이터 사이즈
- SLOW_TIME: 슬로우모션이 되는 시간(단위: sec)
- SLOW_FRAME: 슬로우모션 동안의 프레임 수
- FRAME_PER_SLOW: 슬로우모션을 하기위해 같은 프레임을 뿌리는 수
- SAVE_FILES: 세이브 하는데 사용할 파일의 수
- SAVE_FILE_DURATION: 파일당 저장하는 시간(단위: 초)
- SAVE_DURATION: 총 버퍼의 데이터사이즈
- FRAME_PER_FILE: 파일당 이미지 수
- TIME_PER_FRAME: 한 이미지당 시간(단위: msec)
- SAVE_FRAMES: 버퍼에 저장되는 총 이미지 수

참고: 시간 개념

캡쳐 디바이스로 부터 이미지를 저장하기 전의 시간을 시작 시간으로 하고 캡쳐를 진행하면서 그 시점의 시간을 저장하는데 이를 현재 시간이라고 하자. fps(Frame Per Second)가 5라고 가정하자.



elapse_time=current_time-start_time으로 캡쳐를 처음 시작할 때부터 현재 이미지를 저장할 때 까지의 시간을 나타낸다.

(1)은 시작 시간을 나타낸다.

(2)는 TIME_PER_FRAME 즉, 한 이미지가 저장되고 그 다음 이미지가 저장되는 시간차를 나타낸다. 현재는 200ms이다.
(3)은 첫번째 이미지(frame_no=0)이 저장될 때의 시점
(4)는 두번째 이미지(frame_no=1)이 저장될 때의 시점
(5)는 두번째 이미지를 저장하고 200ms가 지나지 않은 상태에서 다음 이미지를캡쳐하고자 할 때의 시점으로, 이 경우에는 최소한 (6)까지 sleep을 한다. 이렇게 하는 이유는 초당 저장되는 이미지의 개수를 보장하기 위해서이다.
(7)은 어떤 이유에서든지 (6)에서 이미지를 캡쳐하고 기준시간인 200ms보다 오랜 시간 후에 이미지가 캡쳐된 상황이다. (7)의 경우가 발생하면 위에서 가정한 초당 다섯 이미지를 저장하겠다는 가정을 지킬 수가 없게 된다. 따라서 (7) 시점에서는 같은 이미지를 2번 저장하도록 한다.(초당 다섯 이미지를 저장하는 것을 보장하기 위해서) 즉, 캡쳐하지 못한 이미지의 개수만큼 같은 이미지를 저장하여 초당 이미지의 개수를 정확하게 맞추게 한다.

3.2 프로세스별 알고리즘

본 절에서는 5가지 프로세스인 dvrtv-save, dvrtv-play,dvrtv-recorder, dvrtv-decoder, dvrtv-reservation의 자료구조 및 알고리즘에 대해서 기술한다.

3.2.1 dvrtv-save

dvrtv-save 프로세스는 사용자가 미리 지정한 초당 이미지 수만큼 캡쳐된 이미지를 저장할 수 있도록 보장한다.

3.2.1.1 자료구조

- 배열
 . char file_name[100];
  프로세스 실행시 넘어오는 두번째 파라미터를 저장하기 위한 배열이다. 이 두번째 파라미터는 dvrtv-save에서 저장하기 위한 파일명을 사용자로 부터 넘겨 받은 것이다.
 . char number_string[10];
  프로세스 실행시 넘어온 두번째 파라미터를 가지고 여러 개의 파일 이름을 생성하기 위해 01부터 최대 파일 수까지의 숫자 스트링을 가지고 있는 배열. '-'기호 위에 숫자를 붙여 저장하고 있다.
 . unsigned char audio_buffer[audio_buffer_size];
  TV 카드에서 받아온 사운드를 영상의 한 이미지에 해당하는 크기만큼 저장하기 위한 버퍼

- 포인터
 . unsigned long *frame_yuv;
  캡쳐 디바이스로부터 얻어온 이미지(yuv 포맷)의 위치를 가지는 포인터

- 변수

. unsigned long start_time;

dvrtv-save 프로세스를 처음 수행하는 시점의 시간 정보를 가지는 변수

. unsigned long end_time;

이미지를 캡쳐하는 과정을 모두 마칠 때의 시간을 나타내는 변수

. unsigned long current_time;

새로운 이미지를 캡쳐하고자하는 현재 시간을 나타내는 변수

. unsigned long elapse_time;

current_time-start_time으로, 캡쳐를 시작한 이후 부터 현재까지의 시간을 나타내는 변수

. unsigned long frame_no;

초기값을 0으로 하고 현재 캡쳐할 이미지의 번호를 나타내는 변수

. int audio_buffer_size;

영상 한 이미지에 해당하는 사운드의 버퍼의 크기. 위에서 기술한 배열audio_buffer에 값이 적용됨. 그러므로 변수 선언시 초기화도 같이 해야함. 값은 SAMPLE_RATE*CHANNELS*2/FRAMES 가 됨

. int fd;

캡쳐 디바이스로부터 읽어온 이미지가 저장되는 파일의 파일 기술자를 저장하는 변수

. int video_dev;

video4linux 장치파일(보통 /dev/video0)의 파일 기술자를 저장하는 변수

. int audio_dev;

OSS 장치파일(보통 /dev/dsp)의 파일 기술자를 저장하는 변수

. int n;

현재 캡쳐해야할 이미지의 개수를 나타내는 변수

. int signal_on;

채널 변경이 가능한지의 여부를 알려주는 플래그 변수

3.2.1.2 알고리즘

```
dvrtv-save(int argc, char **argv)
{
  /* video4linux을 사용할 수 있도록 기본값을 설정한다.
     argv[1]은 사용하고자하는 캡쳐 장치 파일명(/dev/video0)을 나타냄 */
  video_dev=v4l_open(argv[1]);
  /* video4linux 사운드 켬 */
  v4l_set_audio_mute(video_dev,0);

  /* OSS를 사용하여 사운드를 저장할 준비.
     argv[2]는 OSS 장치 파일명(/dev/dsp)을 나타냄*/
  audio_dev=oss_open(argv[2]);
```

```
  oss_audio_init();

  start_time 변수에 시작 시간을 넣는다;
  for(;;){
    current_time 변수에 현재 시간을 넣는다;
    시간을 확인하여 현재 저장해야할 이미지의 개수를 파악;

    /* 오디오를 OSS로 읽음 */
    read(audio_dev,audio_buffer,audio_buffer_size);

    if(그 시간에 저장해야 할 파일이 있을경우)
      for(i=0; i<n; i++){
        이미지 수를 1개 파일당 이미지 수로 나누어 파일의 번호를 확인 번호에 맞는
        파일을 오픈한다;
        최대 파일 숫자보다 넘어가게 되면 다시 처음 파일로 파일 번호를 바꾸어 주어
        처음 파일을 오픈하게 하고, 파일 포인터를 처음으로 움직인다;
      }
      이미지를 저장;
      읽은 오디오를 영상 한이미지에 해당하는 양만큼 저장;
      이미지 넘버를 증가;
    }else{
      usleep으로 이미지를 저장해야 할 시간까지 기다림;
    }
  }
  close(fd);
}
```

3.2.2 dvrtv-play

dvrtv-play 프로세스는 dvrtv-save 프로세스가 저장해 놓은 파일을 지연없이 플레이 할수 있도록 보장한다. dvrtv-play 프로세스의 메인(main)에서는 window를 생성 후 이벤트가 발생될 때까지 블록 된다. 그로 인해 화면 출력은 쓰레드를 이용해야한다. 그러므로 본 문서에서는 dvrtv-play 프로세스에서 제일 중요한 play()함수를 설계한다.

3.2.2.1 자료구조

- 배열
 . char file_name[100];
  프로세스 실행시 넘어오는 두번째 파라미터를 저장하기 위한 배열이다. 이 두번째 파라미터

는 dvrtv-save에서 저장하기 위한 파일명을 사용자로 부터 넘겨 받은 것이다.

. char number_string[10];

파일 이름에 번호를 붙여주기 위해 사용하는 임시 문자열

. unsigned char frame_yuv[WIDTH*HEIGHT*3/2];

버퍼에서 읽어온 yuv이미지가 저장되는 배열

. unsigned char move_string[15],volume_string[15],channel_string[15];

화면에 출력하기 위한 정보를 가지는 문자열

- 변수

. int fd;

캡쳐 디바이스로 부터 읽어온 이미지가 저장되는 파일의 파일 기술자를 저장하는 변수

. int audio_dev;

파일로 부터 읽어온 사운드 데이터를 OSS 장치 파일로 출력하는데 쓰이는 파일 기술자

. int volume;

현재 volume 값을 가지고 있는 변수

. int channel;

현재 방송채널의 값을 가지고 있는 변수

. int shift_left_count;

타임 쉬프팅에서 현재 시간보다 전으로 이동할 경우 1씩 증가되는 변수

. int shift_right_count;

타임 쉬프팅에서 쉬프팅되어있는 상태에서 현재시간 쪽으로 이동할 경우 1씩 증가되는 변수

. int shift_count;

타임 쉬프팅에서 현재 시간에서 실제로 이동 되어있는 수를 가지고 있는 변수

. int buffer_full_flag;

전체 파일을 한바퀴 돌경우 1로 세팅되어 파일의 처음으로 쉬프팅할 경우 파일의 뒷부분으로 이동할 수 있도록 하는 플래그 변수

. int buffer_file_num;

현재 파일의 숫자를 가지고 있는 변수

. int frame_no;

초기값을 0으로 하고 현재 플레이할 이미지의 번호를 나타내는 변수

. unsigned long start_time;

dvrtv-play 프로세스를 처음 수행하는 시점의 시간 정보를 가지는 변수

. unsigned long end_time;

이미지를 플레이하는 과정을 모두 마칠 때의 시간을 나타내는 변수

. unsigned long current_time;

새로운 이미지를 플레이하고자하는 현재 시간을 나타내는 변수

. unsigned long elapse_time;

current_time-start_time으로, 플레이를 시작한 이후 부터 현재까지의 시간을 나타내는 변수

. int n;

현재 플레이해야할 이미지의 개수를 나타내는 변수

. Widget display, top_shell;

영상 출력을 위한 X 위젯의 ID

. float zoom;

이미지의 확대,축소 비율

. int xv_port;

이미지를 출력하기 위한 포트번호

. XvImage *image;

출력하기 위한 이미지를 저장하는 공유메모리

. int pause_frame_no;

일시정지 되어있는 동안 증가되는 변수로서 일시정지 동안 넘어간 프레임 수를 가지고 있다. 이 변수 값이 JUMP_TIME*FRAME_PER_SEC와 같을 경우 0으로 초기화 된다

. int slow_state;

슬로우 키가 눌린 경우 하나씩 증가하면서 총 slow 시켜야하는 프레임 수 만큼을 체크하는 변수이다.

3.2.2.2 알고리즘

```
void *play(void *data)
{
  XVideo 라이브러리를 사용 할 수 있는지 디바이스 정보를 확인한다;
  디바이스의 출력 포트를 설정한다;
  이미지 출력을 위한 X서버와의 공유 메모리를 설정한다;

  start_time 변수에 시작 시간을 넣는다;

  for(;;){
    if(shift_left_count가 있으면){
      if(현재 파일 포인터의 위치가 점프할 양보다 작으면){
        현재 파일 포인터를 닫고,
        파일 포인터에 전 파일을 열어서 넣고,
        파일 포인터의 위치를 파일의 뒤에서 남은양 만큼 이동;
      }else{
        파일 포인터를 쉬프팅해야하는 양만큼 이동한다;
      }
    }
    if(shift_right_count가 있으면){
      if(현재 파일의 남은 양이 점프할 양보다 작으면){
        현재 파일 포인터를 닫고,
```

```
    파일 포인터에 다음 파일을 열어서 넣고,
    파일 포인터의 위치를 파일의 앞으로 세팅;
  }else{
    파일 포인터를 쉬프팅해야하는 양만큼 이동한다;
  }
}

if(일시정지 프레임 숫자가 JUMP_TIME*FRAME_PER_SEC랑 같으면){
  shift_count를 감소 시킨다;
  일시정지 프레임 숫자를 초기화한다;
}
current_time 변수에 현재 시간을 넣는다;
시간을 확인하여 현재 플레이해야할 이미지의 개수를 파악;

if(그 시간에 플레이해야 할 이미지가 있을경우)
  for(i=0; i<n; i++){
    현재 이미지 수를 파일개당 이미지 수로 나누어 파일의 번호를 확인 번호에 맞는
    파일을 오픈한다;
    최대 파일 숫자보다 넘어가게 되면 다시 처음 파일로 파일 번호를 바꾸어 주어
    처음 파일을 오픈하게 하고, 파일 포인터를 처음으로 움직인다;
    if(i==(n-1)){
      if(슬로우 키가 눌릴 경우){
        if(슬로우 키가 눌린것이 처음일 경우){
          shift_left_count를 하나 증가시킨다;
          키가 눌린것이 처음임을 가르키는 플레그 0으로 초기화;
        }
        슬로우 상태를 증가시킨다;
        if(슬로우 상태가 총 슬로우시킬 프레임과 같으면){
          if(shift_count가 0이 아닐경우){
            shift_left_count를 증가시킨다;
          }
          슬로우 키 플레그를 초기화 시킨다;
          슬로우 상태를 초기화 시킨다;
        }
        if(일시정지 키가 눌리지 않고, 다음의 두 상태중 한개 일때
          1. 슬로우 키가 눌린상태에서 슬로우 상태에 5를 모듈러 연산한 값이 0일 경우
          2. 슬로우 키가 눌리지 않았을 경우){
            영상 한 이미지 읽는다;
            소리 한 프레임 읽는다;
```

```
                한 이미지의 이미지에 출력 정보를 그린다;
                현재 창 크기에 맞춰 이미지 크기를 조정한다;
                X 출력을 위한 공유 메모리에 이미지를 쓴다;
            }else{
              if(일시정지 키가 눌리지 않고, 다음 두 상태중 한개 일때
                1. 슬로우 키가 눌린상태에서 슬로우 상태에 5를 모듈러 연산한 값이 0일 경우
                2. 슬로우 키가 눌리지 않았을 경우){
                  영상 한 이미지 지나간다;
                  소리 한 이미지 읽는다;
              }
            }
          }
          if(일시정지 키가 눌리지 않고, 다음 두 상태중 한개 일때
            1. 슬로우 키가 눌린상태에서 슬로우 상태에 5를 모듈러 연산한 값이 0일 경우
            2. 슬로우 키가 눌리지 않았을 경우){
              소리 한 이미지 쓴다;
              이미지 넘버를 증가;
          }else if(일시정지 키가 눌리지 않고, 다음 두 상태중 한개 일때
            1. 슬로우 키가 눌린상태에서 슬로우 상태에 5를 모듈러 연산한 값이 0이 아닌 경우
            2. 슬로우 키가 눌리지 않았을 경우){
              일시정지 프레임 숫자를 증가;
          }
        }else{
          usleep으로 이미지를 저장해야 할 시간까지 기다림;
        }
      }
      close(fd);
    }

    void dvrtv-decoder(void *param)
    {
      struct filename_group *filename= (struct filename_group *)param;
      AVFormatContext ic;
      AVPacket pkt;
      av_register_all();
      av_open_input_file(&ic, filename->mpegfilename, NULL, 0, NULL);
      for(;;)
      {
        ret=av_read_packet(ic, pkt);
```

```
    if(ret<0)
      break;

    if(pkt->stream_index==video_index)
    {
      영상 데이터 디코딩;
    }
    else if(pkt->stream_index==audio_index)
    {
      소리 데이터 디코딩;
    }
    av_free_packet(pkt);
  }
}
```

3.2.3 dvrtv-recorder

dvrtv-save에서 저장한 파일을 읽어서, ffmpeg 라이브러리를 사용하여 mpeg 포맷으로 저장한다.

3.2.3.1 자료구조

- 포인터
 . AVFormatContext *ic;
  ffmpeg 라이브러리 자료형이다. mpeg파일에 접근하기 위한 모든 정보를 저장한다. av_read_packet()의 첫번째 인자로 넣어주는 포인터
 . AVFrame *picture_ptr
  ffmpeg 라이브러리 자료형이다. 영상 데이터를 저장하여avcodec_encode_video()인자로 넣어, 영상 인코딩을 수행하기 위한 포인터
 . char *video_outbuf;
  avcodec_encode_video()에 인자로 넣어 호출하여, 인코딩한 영상 데이터를 저장하는 포인터

- 변수
 . int out_size
   인코딩한 데이터 size를 나타내는 변수

3.2.3.2 알고리즘

```
dvrtv-recorder(int argc, char **argv)
{
  AVFormatContext *oc;
  AVFrame *picture_ptr;
  char *video_outbuf;
  int out_size;

  ffmpeg 라이브러리를 초기화;

  /* raw파일로 부터 영상과 소리 데이터를 읽어 온다 */
  video = read(f1, video_size);
  audio = read(f1, audio_size);

  /* 영상 데이터를 인코딩 후에 mpeg 포맷으로 저장한다. */
  avcodec_encode_video(c, video_outbuf, video_outbuf_size, picture_ptr);
  av_write_frame(oc, video_index, video_outbuf, out_size);
  /* 소리데이터를 인코딩 후에 mpeg 포맷으로 저장한다. */
  avcodec_encode_audio(...);
  av_write_frame(...);
}
```

3.2.4 dvrtv-reservation

dvrtv-reservation 프로세스는 실행시 넘겨받은 예약정보로 매 분마다 시간을 확인해서 시작 시간과 종료 시간에 맞춰 DVR TV를 실행시키고 종료시킨다.

3.2.4.1 자료구조

- 구조체
 . struct reserve reservation;
   예약정보를 가지는 구조체로서 구조체 멤버 변수는 다음과 같다.

```
struct reserve
{
  struct tm start; // 예약 시작 시간
  struct tm end;    // 예약 종료 시간
  int channel;      // 채널
  char desc[200];   // 설명
}
```

- 배열
 . char res_info[];
   실행시 받은 예약정보를 가지는 배열

- 변수
 . pid_t pid;
   자식 프로세스의 식별번호
- 매크로
 . START
   예약 시작
 . END
   예약 끝

3.2.4.2 알고리즘

dvrtv-reservation(int argc, char **argv)
{
  프로세스 실행시 넘겨받은 예약정보를 시작시간,종료시간,채널,설명으로 나눠 struct reserve
  구조체에 저장한다;

  if(시작시간이 됐는지 확인) {
    switch(pid = fork()) {
    case 0:
      dvrtv 실행;
    default:
      if(종료시간이 됐는지 확인)
        dvrtv 종료;
    }
  }
}

4. 세부 함수 알고리즘

4.1 video4linux 관련 함수

본 절에서는 video4linux 관련 함수를 기술한다.

4.1.1 v4l_open()

캡쳐 디바이스를 사용하기 전에 디바이스를 초기화하는 작업을 한다.
함수 원형은 int v4l_open(const char *file)이다.

4.1.1.1 자료구조

- 구조체

. struct video_capability cap;
  디바이스로 부터 이미지를 얻기 위해 필요한 구조체 video4linux 문서 참조
. struct video_channel chan;
  디바이스로 부터 이미지를 얻기 위해 필요한 구조체 video4linux 문서 참조
. struct video_window win;
  디바이스로 부터 이미지를 얻기 위해 필요한 구조체 video4linux 문서 참조
. struct video_picture pic;
  디바이스로 부터 이미지를 얻기 위해 필요한 구조체 video4linux 문서 참조
. struct video_mbuf m_buf;
  디바이스로 부터 이미지를 얻기 위해 필요한 구조체 video4linux 문서 참조
. static struct video_mmap v_map;
  디바이스로 부터 이미지를 얻기 위해 필요한 구조체 video4linux 문서 참조

- 포인터

. static unsigned char *frame;
  yuv 포맷 이미지를 가진 배열 포인터

- 변수

. static int fd;
  이미지를 얻기 위한 디바이스의 파일 기술자를 저장하는 변수
. int fno;
  더블버퍼링을 하기위한 이미지의 숫자를 가지고 있는 변수

4.1.1.2 알고리즘

위의 구조체를 이용하여 캡쳐 디바이스를 초기화하는 알고리즘을 기술한다.
먼저 구조체 video_capability를 이용하여 캡쳐 디바이스가 제공하는 기능을 파악한 후에, 사용하고자하는 채널 선택 및 설정(구조체 video_channel 이용), 캡쳐 영역 지정(구조체 video_window 이용), 밝기와 채도등을 설정하는 이미지 특성 설정(구조체 video_picture 이용), 버퍼의 크기를 mmap interface에게 알려주기(구조체 video_mbuf 이용), mmap interface를 이용하기(구조체 video_mmap 이용)등의 과정을 거치게 된다. 반환 값은 fd(파일 기술자)이다. 알고리즘은 다음과 같다.

int v4l_open(const char *file)

{
/* cap, chan, win, pic */
각 구조체 선언;

fd=open(VIDEVICE_NAME, O_RDONLY);

VIDIOCGCAP과 구조체 cap을 이용한 ioctl의 호출을 통해 디바이스 정보를 얻어온다;

/* 사용하고자하는 채널 선택 및 원하는 값으로 설정한다. */
chan.channel=0;
chan.flags=0;
chan.type=VIDEO_TYPE_CAMERA;
chan.norm=VIDEO_MODE_NTSC;
VIDIOCSCHAN과 구조체 chan을 이용한 ioctl의 호출을 통해 채널 관련 정보값을 변경한다;

VIDIOCGWIN과 구조체 win을 이용한 ioctl의 호출을 통해 윈도우 관련 정보를 얻어온다;
win.width = WIDTH;
win.height = HEIGHT;
win.x=win.y=win.chromakey=win.flags=0;
VIDIOCSWIN과 구조체 win을 이용한 ioctl의 호출을 통해 윈도우 관련 정보를 변경한다;

VIDIOCGPICT와 구조체 pic을 이용한 ioctl의 호출을 통해 이미지 특성 관련 정보를 얻어온다;
pic.depth=32;
pic.palette=VIDEO_PALETTE_YUV420P;
VIDIOCSPICT와 구조체 pic을 이용한 ioctl의 호출을 통해 이미지 특성 관련 정보를 변경한다;

VIDIOCGBUF와 구조체 m_buf를 이용한 ioctl의 호출을 통해 버퍼 관련 정보를 얻어온다;
frame=(char *)mmap(0, m_buf.size, PROT_READ, MAP_SHARED, fd,0);

v_map.width=WIDTH;
v_map.height=HEIGHT;
v_map.format=VIDEO_PALETTE_YUV420P;
v_map.frame=0;

for(fno=0;fno<m_buf.frames;fno++){
  장치의 최대 버퍼 싸이즈만큼 영상을 캡쳐 하여 버퍼를 초기화 한다;
}

```
  v4l_capture(fd);

  fd 반환;
}
```

4.1.2 v4l_capture()

캡쳐 디바이스를 이용하여 캡쳐를 한다.
함수원형은 unsigned char *v4l_capture(int fd)이다.

4.1.2.1 자료구조

- 구조체
 . struct video_mmap v_map;
   4.1.1.1에서 기술된 구조체 사용

- 포인터
 . unsigned char *frame;
  계속 해서 저장되고 있는 버퍼에서 현재 읽어와야 하는 버퍼의 이미지를 가르키는 포인터

4.1.2.2 알고리즘

본 절에서는 이미지를 캡쳐하는 알고리즘에 대해서 알아본다. 알고리즘은 다음과 같다.

```
unsigned char *v4l_capture(int fd)
{
  v_map의 현재 이미지에 fno를 넣는다.

  VIDIOCSYNC와 구조체 v_map을 이용한 ioctl의 호출을 통해 이미지 캡쳐가 끝날 때까지
  기다린다;

  if(signal_on이 1일 경우){
    공유 메모리로 부터 방송 채널번호를 읽어옴
    가져온 채널번호를 tv card에 설정
  }

  *frame에 m_buf의 offsets[fno]를 통해 실제 메모리 주소를 맵핑한다.

  v_map의 현재 이미지에 fno를 넣는다.
```

VIDIOCMCAPTURE와 구조체 v_map을 이용한 ioctl의 호출을 통해 이미지를 디바이스로부터 얻어온다;

fno를 증가한다.
if(fno가 버퍼의 최대 이미지 수가 되면){
fno를 0으로 설정한다.
}

frame을 반환한다.
}

4.1.3 v4l_set_audio_mute()

TV 카드로 부터 수신되고 있는 음성신호를 켜고 끄는 기능을 수행한다.
함수 원형은 void v4l_set_audio_mute(int fd,int mute);

4.1.3.1 자료구조

- 구조체
. struct video_audio audio;
video4linux에 사운드와 관련된 데이터가 저장되는 구조체

4.1.3.2 알고리즘

앞에서 기술한 v4l_open() 으로 open한 video4linux 장치 파일에 ioctl을 이용하여 사운드 부분을 켜고 끄는 것이 이 함수의 목적이다. 따라서 video4linux에 정의된 구조체(video_audio)와 ioctl을 이용하여 장치의 현재 상태를 받아오고 그것을 가지고 사운드를 켜거나 끄는 부분만 바꾸어 다시 장치에 저장하면 된다. 알고리즘은 다음과 같다.

```
void v4l_set_audio_mute(int fd, int mute)
{
  struct video_audio audio;

  /* video4linux에서 오디오 정보 가져오기 */
  if (ioctl(fd,VIDIOCGAUDIO,&audio) < 0)
      에러 메시지 출력;
  else {
    if(mute가 참이면)
      audio;
```

```
    else
      소리를 켬;
  }
  /* video4linux에 오디오 정보 저장하기 */
  if (ioctl(fd,VIDIOCSAUDIO,&audio) < 0)
    에러 메시지 출력;
}
```

4.1.4 v4l_get_tuner_station()

본 함수는 캡쳐 디바이스로 부터 현재 설정되어 있는 방송국 채널정보를 가져온다. 함수원형은 int v4l_get_tuner_station(int fd)이다. 인수는 캡쳐 디바이스 기술자이며, 방송국 채널정보를 반환한다.

4.1.4.1 자료구조

- 변수
 . unsigned long frequency;
   캡쳐 디바이스로 부터 넘겨받은 주파수가 저장되는 변수

4.1.4.2 알고리즘

본 함수의 알고리즘에 대하여 기술한다. 캡쳐 디바이스는 주파수에 의해 방송국의 채널이 변경된다. 가령 표준으로 정해져 있는 주파수표(frequency table)에 따라 특정 지역의 ch 몇번은 주파수 몇 헤르쯔(Hz)라는 식이다. 따라서 캡쳐 디바이스에서 읽어오는 값은 주파수(Hz)이며 그것을 특정 주파수표에 따라 그에 해당하는 채널 번호를 찾아 값을 반환해야 한다.

```
int v4l_get_tuner_station(int fd)
{
  unsigned long frequency;

  캡쳐 디바이스로 부터 현재의 주파수를 변수 frequency로 가져옴;

  return 주파수표로 부터 변수 frequency에 해당하는 채널 반환;
}
```

4.1.5 v4l_set_tuner_station()

본 함수는 변경하고자 하는 방송국 채널을 인수로 받아 캡쳐 디바이스에 그에 해당하는 주파

수를 적용해준다. 함수원형은 void v4l_set_tuner_station(int fd, int station)이다. 첫번째 인수는 캡쳐 디바이스의 기술자이며, 두번째 인수는 방송국 채널번호이고, 반환값은 void형이다.

4.1.5.1 자료구조

- 변수
 . unsigned long frequency;
  주파수표(frequency table)로 부터 인수로 주어진 방송국 채널에 해당하는 주파수가 저장되는 변수

4.1.5.2 알고리즘

본 함수의 알고리즘에 대하여 기술한다. 본 함수는 위에서 설명한 함수이다.
v4l_get_tuner_station()과 반대로 작동한다.

```
void v4l_set_tuner_station(int fd,int station)
{
  unsigned long frequency;

  frequency = 인수로 받은 방송국 채널번호에 해당하는 주파수표의 주파수;

  /* 캡쳐 디바이스의 소리를 끔 */
  v4l_set_audio(fd,1);

  변수 frequency에 저장된 값을 캡쳐 디바이스에 셋팅;

  /* 캡쳐 디바이스의 소리를 다시 켬 */
  v4l_set_audio(fd,0);
}
```

4.2 OSS 관련 함수

본 절에서는 OSS 관련 함수를 기술한다.

4.2.1 oss_init()

OSS 디바이스를 사용하기 전에 초기화를 한다.
함수 원형은 void oss_init(int fd,int sr,int ch) 이다.

4.2.1.1 자료구조

- 변수

. int af = AFMT_S16_LE;

샘플 포맷을 저장하는 변수

4.2.1.2 알고리즘

이 함수는 OSS 오디오 디바이스 파일인 /dev/dsp 파일에 sample format, sample rate, channel에 관한 것들을 설정한다. sample format을 제외한 나머지 것들은 함수의 인수로 받는다. sample format의 경우 컴퓨터 아키텍쳐에 따라 고정적이므로 함수안에서 직접 설정한다.

```
void oss_init(int fd, int sr, int ch)
{
    int af = AFMT_S16_LE;

    ioctl(fd,SNDCTL_DSP_SETFMT,&af);
    ioctl(fd,SNDCTL_DSP_CHANNELS,&ch);
    ioctl(fd,SNDCTL_DSP_SPEED,&sr);
}
```

4.2.2 oss_mixer_get_volume()

이 함수는 OSS 믹서 디바이스로 부터 현재 볼륨을 넘겨받아 반환한다.
함수 원형은 int oss_mixer_get_volume(void) 이다.

4.2.2.1 자료구조

- 변수

. int fd;

OSS 믹서 디바이스 파일 /dev/mixer의 파일 기술자

. int currunt_volume;

현재 볼륨을 저장

. int devs;

사운드 카드의 믹서가 지원하는 채널의 종류를 저장

4.2.2.2 알고리즘

이 함수는 사운드 믹서에서 특정 채널의 볼륨정보를 알아보고 싶을 때 사용한다. 알고리즘은 다음과 같다.

```
int oss_mixer_get_volume( void )
{
    int fd, devs;
    int current_volume = 0;

    fd = open( "/dev/mixer", O_RDONLY );
    if( fd != -1 ) {
        /* 마스크로 이용하여 믹서의 채널조율를 읽어옴 */
        ioctl( fd, SOUND_MIXER_READ_DEVMASK, &devs );
        if( devs & 설정하고자 하는 믹서 채널의 플래그) {
            ioctl(fd, MIXER_READ(설정하고자 하는 믹서 채널), &current_volume);
            close(fd);
        } else {
            close( fd );
            return current_volume;
        }
  }
  return current_volume;
}
```

4.2.3 oss_mixer_set_volume()

이 함수는 OSS 믹서 디바이스로 부터 설정하고자 하는 볼륨을 설정한다.
함수 원형은 int oss_mixer_set_volume(int percent_diff) 이다.

4.2.3.1 자료구조

- 변수
 . int fd;
   OSS 믹서 디바이스 파일 /dev/mixer의 파일 기술자
 . int devs;
   사운드 카드의 믹서가 지원하는 채널의 종류를 저장
 . int level_percentage;
   현재 볼륨

4.2.3.2 알고리즘

이 함수는 인수로 주어진 수 만큼의 볼륨 증가분을 현재 볼륨에 더한다. 들어온 값을 현재 볼륨에 더하고 그것을 디바이스 파일을 open하여 ioctl로 설정한다. 알고리즘은 다음과 같다.

```
int oss_mixer_set_volume( int percent_diff )
{
    int fd, v, cmd, devs;
    int level_percentage;

    level_percentage = oss_mixer_get_volume();

    level_percentage + = percent_diff;

    if( level_percentage > 100 ) level_percentage = 100;
    if( level_percentage < 0 ) level_percentage = 0;
    fd = open( "/dev/mixer", O_RDONLY );
    if( fd != -1 ) {
        ioctl(fd, SOUND_MIXER_READ_DEVMASK, &devs);
        if(devs & 설정하고자 하는 믹서 채널의 플래그) {
            ioctl(fd, MIXER_WRITE(설정하고자 하는 믹서 채널), &level_percentage);
        } else {
            close(fd);
            return 0;
        }
        close( fd );
        return level_percentage;
     }

     return 0;
}
```

4.3 yuv2rgb 관련함수

본 절에서는 yuv 포맷 이미지를 rgb 포맷 이미지로 변환하는 rgb 관련 함수를 기술한다.

4.3.1 yuv2rgb32()

위에서 말한 바와 같이 yuv 포맷 이미지를 rgb32 포맷으로 바꾼다.
함수원형은 void yuv2rgb32(char *out_addr, char *in_addr, int rowstride, int width, int height)

4.3.1.1 자료구조

- 변수
 . unsigned char *y,*u,*v;
    y,u,v 값의 포인터

4.3.1.2 알고리즘

```
void yuv2rgb32 (char *out_addr, char *in_addr, int rowstride, int width, int height)
{
  rgb32 포맷을 위한 초기화와 할당;
  y,u,v 각각에 해당 in_addr을 지정한다;

  yuv를 rgb32로 고침;
  rgb32데이터를 out_addr에 전달;
}
```

4.4 공유메모리 관련 함수

본 절에서는 공유메모리(shm:shared memory) 관련 함수를 기술한다.
공유메모리는 프로세스간 공유되는 정보(프로세스 채널정보, 방송국 채널번호 등)를 쓰고 읽기 위해 사용한다.

4.4.1 shm_write()

본 함수는 사용하는 공유메모리를 설정하고 인수로 받은 값을 메모리에 쓴다.
함수원형은 int shm_write(key_t key,size_t n,const char* data) 이다.
첫번째 인수는 공유메모리의 이름(수)이고, 두번째 인수는 설정할 메모리의 크기이며, 세번째는 저장할 데이터의 포인터이다. 반환값은 성공할 경우 공유메모리의 기술자가 반환되고, 실패할 경우 -1을 반환한다.

4.4.1.1 자료구조

- 변수
 . int seg_id;
    공유메모리의 기술자

- 포인터
  . char *mem_ptr;
     공유메모리의 포인터

4.4.1.2 알고리즘

이 함수의 알고리즘의 대하여 기술한다. 본 함수는 공유메모리를 설정하고 그 자리에 주어진 데이터를 쓴다.

```
int shm_write(key_t key,size_t n,const char* data)
{
  int seg_id;
  char *mem_ptr;

  /* 공유메모리 생성 */
  seg_id = shmget(key,n,IPC_CREAT|0766);
  if(seg_id == -1)
    return -1;

  /* 공유메모리의 첫번째 주소를 mem_ptr에 저장 */
  mem_ptr = shmat(seg_id,NULL,0);
  if(mem_ptr == -1)
    return -1;
  /* 공유메모리에 데이터 저장 */
  strcpy(mem_ptr,data);

  return seg_id;
}
```

4.4.2 shm_read()

본 함수는 값이 저장되어 있는 공유메모리로 부터 값을 읽어온다.
함수원형은 char* shm_read(key_t key,size_t n) 이다. 첫번째 인수는 공유메모리의 이름이고, 두번째 인수는 읽어올 데이터 크기(단위 byte)이다. 반환값은 읽은 데이터의 포인터이다.

4.4.2.1 자료구조

- 변수
 . int seg_id;
   공유메모리의 기술자

- 포인터
  . char *mem_ptr;

공유메모리의 포인터

4.4.2.2 알고리즘

이 함수의 알고리즘의 대하여 기술한다. 본 함수는 공유메모리에 설정되어 있는값의 포인터를 반환한다.

```
char* shm_read(key_t key,size_t n)
{
  int seg_id;
  char *mem_ptr;
   /* 이름이 key인 공유메모리를 n byte 만큼 open */
  seg_id = shmget(key,n,0766);
  if(seg_id == -1)
    에러메시지 출력;

  /* 공유메모리에 데이터 저장 */
  mem_ptr = shmat(seg_id,NULL,0);
  if(mem_ptr == -1)
    에러메시지 출력;

  return (char*)mem_ptr;
}
```

4.5 정보 출력 관련함수

출력 화면에 정보를 출력하기 위한 함수를 기술한다.

4.5.1 draw_text()

문자열을 입력받아 정해진 좌표에 출력하는 함수이다.
함수원형은 int draw_text(unsigned char *frame, int startx, int starty, const char *text, int type)

4.5.1.1 자료구조

- 배열
 . struct char_font font_table[];
   아스키 코드와 해당 코드의 출력 폰트를 저장하는 구조체의 배열

- 구조체

. struct char_font;

아스키 코드와 해당 코드의 출력 폰트를 저장하는 구조체

- 변수

. int pos;

문자열의 인덱스

. int x, y;

출력 폰트의 x, y 좌표

4.5.1.2 알고리즘

원본 이미지의 해당 좌표의 픽셀 값을 출력 폰트가 1이면 0으로 바꾼다.

```
int draw_text(unsigned char *frame, int startx, int starty, const char *text, int type);
{
  폰트 출력 위치가 이미지를 벗어나면 위치를 조절한다;
  while(문자열이 남았으면){
    출력 좌표가 이미지 영역 밖이면 break;
    font_table에서 출력할 글자와 같은 글자를 찾는다;
    /* 테두리 */
    for(x=0;x<8;x++){
      for(y=0;y<9;y++){
        if(font_table[i].font[y][x]){
          픽셀의 색을 0으로 바꾼다;
        }
      }
    }
    /* 내부 */
    for(x=0;x<8;x++){
      for(y=0;y<9;y++){
        if(font_table[i].font[y][x]){
          픽셀의 색을 0으로 바꾼다;
        }
      }
    }
  }
}
```

4.6 예약 관련함수

예약에 관한 정보를 처리하기 위한 함수를 기술한다.

4.6.1 parseTime()

문자열로 입력받은 예약정보를 시작시간, 종료시간, 채널, 설명을 나눠 struct reserve 구조체에 저장한다.
함수 원형은 void parseTime(struct reserve *reservation, char *res_info)이다

4.6.1.1 자료구조

- 배열
 . char temp[];
   문자열로 받은 예약 정보에서 필요한 정보만 임시로 저장하는 배열

4.6.1.2 알고리즘

예약정보는 일정한 규칙을 가지고 입력되므로 문자열을 정보에 따라 일정양으로 잘라 구조체에 저장한다. 알고리즘은 다음과 같다

```
void parseTime(struct reserve *reservation, char *res_info)
{
  /* 시작 시간의 년도 을 분리한다. */
  strncpy(temp, res_info, 4); temp[4] = '₩0';
  /* 예약정보 구조체의 해당 영역에 저장한다. */
  reservation->start.tm.year = atoi(temp)-1900;

  위와 같은 방법으로 시작시간의 월,일,시간,분을 저장한다;
  위와 같은 방법으로 종료시간의 년,월,일,시간,분을 저장한다;
  위와 같은 방법으로 채널을 저장한다;

  나머지 설명 부분을 구조체에 저장한다;
}
```

4.6.2 timeCheck()

매 분마다 현재 시간과 예약 시간을 비교해서 같은 시간이 되면 0을 반환하고, 현재 시간이 이미 예약 시간을 지났을 경우 -1을 반환한다.

함수 원형은 int timeCheck(int work)이다.

4.6.2.1 자료구조

- 포인터
. struct tm *now;
  현재 시간을 저장하는 포인터
. struct tm *target;
  예약 시간을 저장하는 포인터

- 변수
. time_t checktime;
  현재 시간을 받는 변수

4.6.2.2 알고리즘

현재 시간과 예약 시간을 비교해서 예약 시간이면 0을 반환하고 아직 아니면 다음 분까지 기다린다. 알고리즘은 다음과 같다.

```
int timeCheck(struct reserve *reservation, int work)
{
  while(1)
  {
    work를 확인해서 시작시간이면 target = &(reservation->start);
                    종료시간이면 target = &(reservation->end);

    /* 현재 시간을 구한다. */
    checktime = time((time_t *) 0);
    now = localtime(&checktime);

    예약 시간과 현재 시간을 비교해서 같으면 0 반환하고
    현재 시간이 예약 시간을 지났으면 -1 반환한다;
    아직 예약 시간이 안됐으면 sleep(60 - now->tm_sec);
  }
}
```

4.7 키 입력 함수

키 입력을 처리하기 위한 함수

4.7.1 keyInput()

X윈도우에서 키 입력을 처리하기 위한 핸들러 함수.
함수 원형은 XtCallbackProc keyInput(Widget w, XtPointer client, XtPointer call_data)이다.

4.7.1.1 자료구조

- 포인터
 . XKeyPressedEvent *cbs;
   핸들러 호출시 키 상태가 저장되는 구조체의 포인터

4.7.1.2 알고리즘

입력된 키와 펑션키의 상태에 따라 해당되는 작업을 수행한다.

```
XtCallbackProc keyInput(Widget w, XtPointer client, XtPointer call_data)
{
  switch(눌려진 키){
  case 'Q': 프로그램 종료;
  case 왼쪽화살표:
    switch(펑션키 상태){
    case 키만 눌렸을때: 채널 감소;
    case 쉬프트 키: 타임 쉬프팅 뒤로;
  }
  case 오른쪽화살표:
    switch(펑션키 상태){
    case 키만 눌렸을때: 채널 증가;
    case 쉬프트 키: 타임 쉬프팅 앞으로;
  }
  case 위화살표:
    switch(펑션키 상태){
    case 키만 눌렸을때: 볼륨 증가;
    }
  case 아래화살표:
    switch(펑션키 상태)
        case 키만 눌렸을때: 볼륨 감소;
  }
}
```

5. 참고문서

[1] ffmpeg Multimedia System, http://ffmpeg.sourceforge.net

[2] Motif Programming Manual, http://www.ist.co.uk/motif/download/6A/6A_book.pdf

[3] Motif Reference Manual, http://www.ist.co.uk/motif/download/6B/6B_book.pdf

[4] Open Sound System Programming's Guide, http://www.opensound.com/pguide/oss.pdf

[5] W.Richard Stevens, "UNIX Network Programing Volume2(Interprocess Communications)", Prentice Hall, 2002.

[6] xfree86 Manual page, http://xfree86.org/4.4.0/manindex3.html

[7] XVideo 예제 프로그램, testxv.c, http://bellet.info/XVideo/testxv.c

[8] 김충석, "X 윈도우시스템 프로그래밍", 이한 출판사, 1994

제2절 DVR Motion 설계

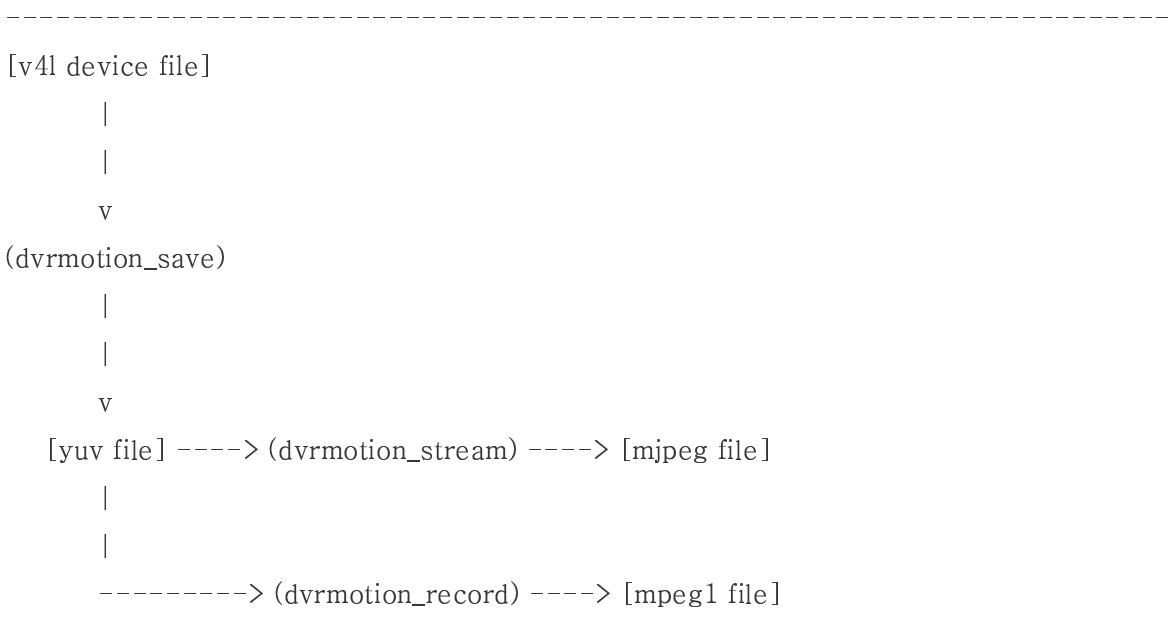
본 절에서는 DVR Motion의 구조 및 기능과 프로세스별 자료 흐름, 프로세스별 및 세부 함수 알고리즘에 관해 기술한다.

1. DVR Motion 구조 및 기능

DVR Motion 부분의 전체적인 구조를 그림으로 표현하고, DVR Motion의 기능을 기술한다. 구조를 그림으로 표시할 때 자료(data), 프로세스, 자료 흐름의 표현은 다음과 같다.

자료: [data name]
프로세스: (process name)
자료 흐름: 화살표(->)

1.1 DVR Motion 구조



1.2 DVR Motion 기능

본 프로젝트에서 DVR Motion 부분은 다음의 기능을 가지고 있다.

- 웹 스트리밍(Web Streaming)
- 모션 검출 기능
- 모션 기록 기능

- 기록 기능

2. 프로세스별 자료 흐름

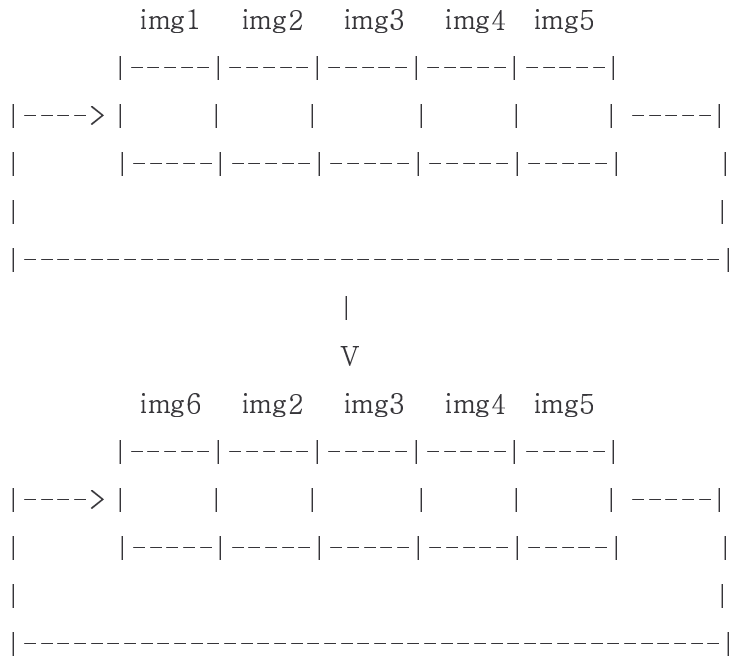
여기에서는 DVR Motion 부분의 핵심 프로세스들의 자료 흐름에 대해서 설명한다. 프로그램을 수행시킬 때, 생성되는 프로세스는 모두 4가지로 dvrmotion-save, dvrmotion-stream, dvrmotion-record, dvrmotion-autorm 등이 있다.

2.1 dvrmotion-save

```
-------------------------------------------------------------------
[v4l device file] ----> (dvrmotion_save) ----> [yuv file]
-------------------------------------------------------------------
```

dvrmotion-save 프로세스는 캡쳐 디바이스인 USB Camera로 부터 이미지(v4l device file)를 읽어오고 그 이미지를 가지고 모션을 검출한다. 모션이 발생했는지를 체크한 후에 캡쳐된 이미지는 yuv 포맷의 raw data로 파일에 쓴다. 이곳에서 사용되는 파일은 캡쳐되는 이미지를 임시 저장하는 파일로 queue처럼 사용된다. 즉, 처음부터 파일을 쓰기 시작해서 파일의 마지막에 이르면 다시 파일의 처음 부분부터 캡쳐된 이미지를 저장하게 된다. 이미지를 캡쳐해서 저장하기까지 초당 이미지 수를 지정하여 매초마다 초당 이미지수만큼 임시 저장하는 파일(yuv file)에 저장한다.

예를 들어 fps(frames per second)가 5라고 가정하고 1초 분량의 이미지를 저장한다고 가정한다.



2.2 dvrmotion-stream

```
-------------------------------------------------------------------
   [yuv file] ----> (dvrmotion_stream) ----> [mjpeg file]
-------------------------------------------------------------------
```

dvrmotion-stream 프로세스는 Storage Device에 저장되어 있는 yuv 포맷의raw data(yuv file)를 읽어서 yuv 포맷 이미지를 jpeg 포맷 이미지로 인코딩한 후 mjpeg 포맷 이미지(mjpeg file)로 전환하여 웹으로 스트리밍을 한다. 이 때 웹 스트리밍은 클라이언트가 요청을 해야만 서비스를 받을 수 있다. dvrmotion-stream 프로세스도 dvrmotion-save 프로세스와 같이 매 초마다 초당 이미지 수 만큼 웹으로 이미지를 보여준다.

2.3 dvrmotion-record

```
-------------------------------------------------------------------
[yuv file] ----> (dvrmotion_record) ----> [mpeg1 file]
                                      |
                                      |
                                      ----> [mpeg1 file]
-------------------------------------------------------------------
```

dvrmotion-record 프로세스는 Storage Device에 저장되어 있는 yuv 포맷의raw data(yuv file)를 읽어서 yuv 포맷의 여러 이미지를 mpeg 포맷으로 변환하여 저장한다. 이 때 그 이미지가 모션이 검출된 이미지인지 검출되지 않은 이미지인지 검사하여 모션 검출된 이미지만 따로 저장하고 전체 이미지도 저장한다.

3. 프로세스별 알고리즘

3.1 전체 자료구조

3.1.1 상수

- WIDTH: 캡쳐된 이미지의 가로 길이(320)
- HEIGHT: 캡쳐된 이미지의 세로 길이(240)
- FRAME_PER_SEC: 초당 캡쳐하고자 할 이미지의 개수(5)
- JPEG_QUALITY: jpeg으로 포맷 변환을 할 때, jpeg 이미지의 상태(quality)(80)
- MOTION: 여러 개의 모션 검출 알고리즘 중에서 하나를 지정할 때 사용(three_frame_diff)
- MOTION_THRESHOLD: 모션이 발생했는지를 알아볼 때 사용하는 임계값(50)
- MOTION_AREA: 모션이 발생했다고 판명된 픽셀의 개수(50)

- ALPHA: 모션 검출 알고리즘에서 배경을 업데이트할 때 사용되는 가중치(0.2)
- SAVE_DURATION: 캡쳐된 이미지를 yuv 포맷의 raw data형식으로 저장하는 시간(60)
- RECORD_DURATION: yuv 포맷의 raw data에서 읽은 이미지를 통해서 mpeg파일로 저장하는시간(SAVE_DURATION*30, 즉 30분 동안 저장하겠다는 뜻)
- MOTION_DURATION: yuv 포맷의 raw data에서 읽은 이미지 중에서 모션이발생한 이미지들을 사용하여 mpeg 파일로 저장하는 시간(SAVE_DURATION*10, 즉 10분 동안 저장하겠다는 뜻)
- SAVE_FRAMES: 캡쳐된 이미지를 저장하는 파일이 새로운 이미지를 덮어쓰지 않고 저장 할 있는 이미지의 개수(SAVE_DURATION*FRAME_PER_SEC)
- RECORD_FRAMES: 특정 시간동안 저장된 이미지의 개수 (RECORD_DURATION*FRAME_PER_SEC)
- MOTION_FRAMES: 특정 시간동안 모션이 발생한 이미지의 개수 (MOTION_DURATION*FRAME_PER_SEC)
- TIME_PER_FRAME: 이미지간의 시간 간격(1000/FRAME_PER_SEC)
- SAVE_LENGTH: yuv 포맷의 raw data를 저장하는 파일의 크기, 즉 버퍼처럼사용되는 파일의 크기(SAVE_FRAME*(WIDTH*HEIGHT*3/2))
- MOTION_NO_MAGIC: 캡쳐된 이미지에 모션이 발생하지 않음을 나타냄(126)
- MOTION_YES_MAGIC: 캡쳐된 이미지에 모션이 발생하였음을 나타냄(127)

참고: 시간 개념은 1절의 DVR TV 설계를 참조

3.2 프로세스별 알고리즘

본 절에서는 4가지 프로세스인 dvrmotion-save, dvrmotion-stream, dvrmotion-record, dvrmotion-autorun의 자료구조 및 알고리즘에 대해서 기술한다.

3.2.1 dvrmotion-save

dvrmotion-save 프로세스는 사용자가 미리 지정한 초당 이미지 수만큼 캡쳐된 이미지를 저장할 수 있도록 보장한다.

3.2.1.1 자료구조

- 포인터
. unsigned long *frame_yuv;
  캡쳐 디바이스로 부터 얻어온 이미지의 위치를 가지는 포인터
- 변수
. unsigned long start_time;
  dvrmotion-save 프로세스를 처음 수행하는 시점의 시간 정보를 가지는 변수

. unsigned long end_time;
dvrmmotion-save 프로세스를 마칠 때의 시간 정보를 나타내는 변수
. unsigned long current_time;
새로운 이미지를 캡쳐하고자하는 현재 시간 정보를 나타내는 변수
. unsigned long elapse_time;
current_time-start_time으로, 캡쳐를 시작한 이후 부터 현재까지의 시간 정보를 나타내는 변수
. unsigned long frame_no;
초기값을 0으로 하고 현재 캡쳐할 이미지 이미지의 번호를 나타내는 변수
. int fd1;
캡쳐 디바이스의 파일 기술자를 저장하는 변수
. int fd2;
캡쳐 디바이스로부터 읽어온 이미지가 저장되는 파일의 파일 기술자를 저장하는 변수
. int n;
현재 캡쳐해야할 이미지의 개수를 나타내는 변수

3.2.1.2 알고리즘

dvrmotion-save 프로세스는 v4l을 사용하기 위한 초기화 작업을 수행한 후, 저장할 파일을 오픈한다. 현재 캡쳐해야할 이미지의 수를 계산하고 캡쳐를 시작한다.

```
dvrmotion-save(int argc, char *argv[])
{
  /* v4l을 사용할 수 있도록 기본값을 설정한다.
      argv[1]은 사용하고자하는 캡쳐 디바이스의 파일명(/dev/video0)을 나타냄 */
  fd1=v4l_open(argv[1]);
  /* argv[2]는 캡쳐 디바이스로 부터 읽어온 이미지가 저장될 파일을 나타냄 */
  fd2=open(argv[2], O_WRONLY|O_CREAT|O_TRUNC,0644);
  start_time 변수에 프로세스 시작 시간을 넣는다;
  for(;;){
    current_time 변수에 현재 시간을 넣는다;
    시간을 확인하여 현재 저장해야할 이미지의 개수를 파악;

    if(n>0){
      이미지를 저장;
      모션 검출 알고리즘을 이용하여 현재 이미지에 모션이 발생했는지를 파악;
      draw_text()
      for(i=0;i<n;i++){
        if(frame_no%SAVE_FRAMES==0)
```

```
            queue로 쓰는 파일의 첫번째 위치로 이동;
          if(write(fd2,frame_yuv,WIDTH*HEIGHT*3/2)!=WIDTH*HEIGHT*3/2){
            fprintf(stderr,"write: %s₩n",strerror(errno));
            exit(1);
          }
        frame_no++;
        }
      } else {
        usleep(TIME_PER_FRAME*(frame_no+1)-elapse_time);
      }
    }
    close(fd2);
}
```

3.2.2 dvrmotion-stream

dvrmotion-stream 프로세스는 dvrmotion-save 프로세스가 저장한 yuv 포맷 이미지를 jpeg 포맷 이미지로 변환하여 클라이언트가 요구를 했을 때, 웹으로 스트리밍하는 기능을 한다.

3.2.2.1 자료구조

- 배열
 . unsigned char frame_yuv[WIDTH*HEIGHT*3/2];
   yuv 포맷 이미지를 저장하기 위한 배열
 . unsigned char frame_jpeg[WIDTH*HEIGHT*3];
   jpeg 포맷 이미지를 저장하기 위한 배열

- 구조체
 . struct sigaction act;
   시그널 처리를 해주기 위해서 사용하는 구조체
 . struct sockaddr_in server;
   스트리밍을 하기 위해서 서버의 주소 정보를 저장할 구조체
 . struct sockaddr_in client;
   스트리밍을 받기 위한 클라이언트의 주소 정보를 저장할 구조체

- 포인터
 . FILE *logfp;
   로그 파일을 저장할 파일 포인터
 . FILE *webin;

연결된 클라이언트를 가리키는 파일 포인터 //읽기 전용
. FILE *webout;
연결된 클라이언트를 가리키는 파일 포인터 //쓰기 전용
. char *infile;
yuv 포맷의 raw data를 가지고 있는 파일을 가리키는 파일 포인터

- 변수
. unsigned long start_time;
스트리밍을 시작할 때의 시간을 저장하는 변수

- 매크로
. STATUS_OK
"HTTP/1.0 200 OK₩n"
. STATUS_NOT_FOUND
"HTTP/1.0 404 Not Found₩n"
. CONTENT_MULTI
"Content-type: multipart/x-mixed-replace;boundary=BoundaryString₩n₩n"
. CONTENT_JPEG
"Content-type: image/jpeg₩n""
. BOUNDARY_STRING
"--BoundaryString₩n"
. END_STRING
"--BoundaryString--₩n"

3.2.2.2 알고리즘

dvrmotion-stream 프로세스는 dvrmotion-save 프로세스가 저장한 yuv 포맷의 이미지를 읽어서 jpeg 포맷의 이미지로 변환하여 클라이언트에게 스트리밍을 하는 기능을 한다.

dvrmotion-stream(int argc, char *argv[])
{
시그널 처리를 하도록 준비;
소켓 생성;
서버의 waiting queue를 설정;
로그파일 생성;
시작 시간 저장;
날짜 정보 저장;
날짜 정보를 이용하여 로그파일의 내용 채움;
while(1){

```
    클라이언트의 요청을 받아들임;
    프로세스 포크함;
    jpeg 포맷의 이미지 이미지 전송;
    부가 정보 전송;
  }
  소켓 연결 끊기;
}
```

3.2.3 dvrmotion-record

dvrmotion-record 프로세스는 yuv 포맷의 이미지를 mpeg 파일로 저장하는 역할을 한다.

3.2.3.1 자료구조

- 배열
 . unsigned char frame_yuv[WIDTH*HEIGHT*3/2];
    yuv 포맷의 이미지를 저장하기 위한 배열

- 포인터
 . FILE *f;
    mpeg 포맷으로 저장할 때 사용되는 파일 포인터
 . FILE *mf;
    모션이 발생한 이미지만 mpeg 포맷으로 저장할 때 사용되는 파일 포인터
 . char *path;
    저장할 파일의 파일명을 가리키는 포인터
 . struct mpeg *mpeg;
    mpeg 포맷으로 저장할 때 사용된다.
 . struct mpeg *mpeg_motion;
    모션이 발생한 이미지만 mpeg 포맷으로 저장할 때 사용된다

- 변수
 . long start_time;
    dvrmotion-record 프로세스가 시작한 때의 시간 정보를 가지는 변수
 . long end_time;
    dvrmotion-record 프로세스를 끝낼 때의 시간 정보를 가지는 변수
 . long elapse_time;
  처음 dvrmotion-record 프로세스를 시작한 때부터 현재 이미지를 mpeg 포맷으로 저장하려
  고 할 때 까지의 경과 시간을 나타내는 변수
 . long current_time;

현재 시간 정보를 가지는 변수

. long frame_no;

mpeg 포맷으로 저장될 이미지의 번호를 가리키는 변수

. long max_frame_no;

최대 mpeg 포맷으로 저장될 이미지의 개수를 가리키는 변수

. long motion_max_frame_no;

최대 mpeg 포맷으로 저장될 모션이 발생한 이미지의 개수를 가리키는 변수

. int mpeg_flag;

모션이 발생하지 않은 이미지도 mpeg 포맷으로 저장할 때, mpeg 파일이 기존에 생성됐는지 아닌지를 나타내는 변수

. int mpeg_motion_flag;

모션이 발생한 이미지를 mpeg 포맷으로 저장할 때, mpeg 파일이 기존에 생성됐는지 아닌지를 나태내는 변수

. int motion_flag;

저장할 이미지에 모션이 포함됐는지 아닌지를 나타내는 변수(0)

. int n;

특정 시점에서 mpeg 포맷으로 저장해야할 이미지의 개수를 나타내는 변수

3.2.3.2 알고리즘

dvrmotion-record 프로세스는 dvrmotion-save 프로세스가 저장한 yuv 포맷의 이미지를 읽어서 mpeg 포맷의 파일로 저장하는 기능을 한다.

```
dvrmotion-record(int argc, char *argv[])
{
  // argv[1]은 입력 파일
  fd=open(argv[1],O_RDONLY);
  /* ffmpeg에서 제공하는 함수들을 사용하기 위해 초기화 한다. */
  avcodec_init();
  avcodec_register_all();
  mpeg 포맷으로 저장하는 시작 시간을 저장;
  for(;;){
    현재 시간을 저장;
    시간을 확인하여 현재 저장해야할 이미지의 개수를 파악;
    if(n>0){
      for(i=0;i<n;i++){
        if(frame_no%SAVE_FRAMES==0)
          lseek(fd,0,SEEK_SET);
```

```
    if(i==(n-1)){
      yuv 포맷의 이미지를 읽어오기;

      if(frame_yuv[0]==MOTION_YES_MAGIC)
        motion_flag=1;
      else
        motion_flag=0;

      if(mpeg_flag==0){
        mpeg 포맷을 저장할 파일을 생성;
        /* mpeg 포맷으로 저장할 수 있도록 기본값 설정 */
        mpeg=mpeg_start(frame_yuv,f,WIDTH,HEIGHT,400000);
        mpeg_flag=1;
      }
      if(mpeg_motion_flag==0 && motion_flag==1){
        모션이 포함된 이미지만 mpeg 포맷을 저장할 파일을 생성;
       /*  mpeg 포맷으로 저장할 수 있도록 기본값 설정 */
       mpeg_motion=mpeg_start(frame_yuv,mf,WIDTH,HEIGHT,400000);
        mpeg_motion_flag=1;
      }

      mpeg_put=(mpeg,frame_yuv);
      if(mpeg_motion_flag==1 && motion_flag==1)
        mpeg_put=(mpeg_motion,frame_yuv);

      if((frame_no+1)%max_frame_no==0){
        mpeg_end(mpeg);
        mpeg_flag=0;
      }
      if(((frame_no+1)%motion_max_frame_no==0)&&
            mpeg_motion_flag==1){
          mpeg_end(mpeg_motion);
          mpeg_motion_flag=0;
      }
    } else
      lseek(fd,WIDTH*HEIGHT*3/2,SEEK_CUR);
    frame_no++;
  }
} else {
```

```
        usleep(TIME_PER_FRAME*(frame_no+ 1)-elapse_time);
      }
    }
    close(fd);
}
```

3.2.4 dvrmotion-autorm

dvrmotion-autorm 프로세스는 쉘 스크립트로서, mpeg 포맷의 파일이 저장되는 디렉토리가 포함된 파일 시스템의 용량을 점검한다. 만약 현재 저장된 파일들의 전체 크기가 파일 시스템의 90% 정도이면 가장 오래전에 저장된 파일을 찾아서 자동으로 지우는 역할을 한다.

4. 세부 함수 알고리즘

4.1 v4l 관련 함수

본 절에서는 v4l 관련 함수를 기술한다.

4.1.1 v4l_open()

캡쳐 디바이스를 사용하기 전에 디바이스를 초기화하는 작업을 한다.
함수 원형은 int v4l_open(const char *file)이다.

4.1.1.1 자료구조

- 구조체
. struct video_capability cap;
  디바이스로 부터 이미지를 얻기 위해 필요한 구조체 v4l 문서 참조
. struct video_channel chan;
  디바이스로 부터 이미지를 얻기 위해 필요한 구조체 v4l 문서 참조
. struct video_window win;
  디바이스로 부터 이미지를 얻기 위해 필요한 구조체 v4l 문서 참조
. struct video_picture pic;
  디바이스로 부터 이미지를 얻기 위해 필요한 구조체 v4l 문서 참조
. struct video_mbuf m_buf;
  디바이스로 부터 이미지를 얻기 위해 필요한 구조체 v4l 문서 참조
. static struct video_mmap v_map;
  디바이스로 부터 이미지를 얻기 위해 필요한 구조체 v4l 문서 참조

- 포인터

. static unsigned char *frame;

yuv 포맷 이미지를 가진 배열 포인터

- 변수

. static int fd;

이미지를 얻기 위한 디바이스의 파일 기술자를 저장하는 변수

4.1.1.2 알고리즘

위의 구조체를 이용하여 캡쳐 디바이스를 초기화하는 알고리즘을 기술한다. 먼저 구조체 video_capability를 이용하여 캡쳐 디바이스가 제공하는 기능을 파악한 후에, 사용하고자하는 채널 선택 및 설정(video_channel 이용), 캡쳐 영역 지정(video_window 이용), 밝기와 채도등을 설정하는 이미지 특성 설정(video_picture 이용), 버퍼의 크기를 mmap interface에게 알려주기(video_mbuf 이용), mmap interface를 이용하기(video_mmap 이용) 등의 과정을 거치게 된다. 반환 값은 fd(파일 기술자) 이다. 알고리즘은 다음과 같다.

```
int v4l_open(const char *file)
{
  /* cap, chan, win, pic */
  각 구조체 선언;
  fd=open(VIDEVICE_NAME, O_RDONLY);
  VIDIOCGCAP과 구조체 cap을 이용한 ioctl의 호출을 통해 디바이스 정보를 얻어온다;
  /* 사용하고자하는 채널 선택 및 원하는 값으로 설정한다. */
  chan.channel=0;
  chan.flags=0;
  chan.type=VIDEO_TYPE_CAMERA;
  chan.norm=VIDEO_MODE_NTSC;
  VIDIOCSCHAN과 구조체 chan을 이용한 ioctl의 호출을 통해 채널 관련 정보값을 변경한다;
  VIDIOCGWIN과 구조체 win을 이용한 ioctl의 호출을 통해 윈도우 관련 정보를 얻어온다;
  win.width = WIDTH;
  win.height = HEIGHT;
  win.x=win.y=win.chromakey=win.flags=0;
  VIDIOCSWIN과 구조체 win을 이용한 ioctl의 호출을 통해 윈도우 관련 정보를 변경한다;
  VIDIOCGPICT와 구조체 pic을 이용한 ioctl의 호출을 통해 이미지 특성 관련 정보를 얻어온다;
  pic.depth=32;
  pic.palette=VIDEO_PALETTE_YUV420P;
  VIDIOCSPICT와 구조체 pic을 이용한 ioctl의 호출을 통해 이미지 특성 관련 정보를 변경한
```

다;
VIDIOCGBUF와 구조체 m_buf를 이용한 ioctl의 호출을 통해 버퍼 관련 정보를 얻어온다;
frame=(char *)mmap(0, m_buf.size, PROT_READ, MAP_SHARED, fd,0);
v_map.width=WIDTH;
v_map.height=HEIGHT;
v_map.format=VIDEO_PALETTE_YUV420P;
v_map.frame=0;

v4l_capture(fd);
fd 반환;
}

4.1.2 v4l_capture()

캡쳐 디바이스를 이용하여 캡쳐를 한다.
함수원형은 unsigned char *v4l_capture(int fd)이다.

4.1.2.1 자료구조

- 구조체
. struct video_mmap v_map;
4.1.1.1에서 기술된 구조체 사용

4.1.2.2 알고리즘

본 절에서는 이미지를 캡쳐하는 알고리즘에 대해서 알아본다. 알고리즘은 다음과 같다.

unsigned char *v4l_capture(int fd)
{
VIDIOCMCAPTURE와 구조체 v_map을 이용한 ioctl의 호출을 통해 이미지를 디바이스로부터 얻어온다;
VIDIOCSYNC와 구조체 v_map을 이용한 ioctl의 호출을 통해 이미지 캡쳐가 끝날 때까지 기다린다;
frame을 반환한다;
}

4.2 모션 검출 알고리즘
본 절에서는 모션 검출 알고리즘의 자료구조와 알고리즘에 대해 기술한다.

4.2.1 three_frame_diff()

모션 검출 알고리즘 중 이미지 3개의 픽셀값의 차를 이용하여 모션을 검출하는 알고리즘에 대해서 기술한다.

4.2.1.1 자료구조

- 배열

. unsigned char frame_tmp[3][WIDTH*HEIGHT];
현재 캡쳐된 이미지의 정보를 저장한다. yuv 포맷 중 y 성분만을 가지고 모션의 발생유무를 알아내기 때문에 WIDTH*HEIGHT 만큼의 크기를 가진다. 현재 이미지와 한 단계 이전 이미지과 두 단계 이전 이미지 값을 저장하는 배열

- 변수

. int init;
초기화가 됐는지를 나타내는 변수(0)
. int n;
현재 이미지를 가리키는 변수(2)
. int n_1;
한 단계 이전 이미지를 가리키는 변수(1)
. int n_2;
두 단계 이전 이미지를 가리키는 변수(0)
. int minx;
모션이 발생한 영역의 최소 픽셀의 x 좌표를 가리키는 변수
. int miny;
모션이 발생한 영역의 최소 픽셀의 y 좌표를 가리키는 변수
. int maxx;
모션이 발생한 영역의 최대 픽셀의 x 좌표를 가리키는 변수
. int maxy;
모션이 발생한 영역의 최대 픽셀의 y 좌표를 가리키는 변수
. int motion;
한 이미지 내에 모션이 발생한 픽셀의 수를 값으로 가지는 변수

4.2.1.2 알고리즘

세 이미지의 픽셀 차이를 가지고 모션 검출을 한다. 현재 이미지와 한 단계 이전 이미지와의 차가 임계값(THRESHOLD)보다 크고, 현재 이미지와 두 단계 이미지와의 차가 임계값보다 크면 모션이 발생한 픽셀의 수를 가지는 변수의 값을 증가시키고 모션이 발생한 픽셀의 수가 지정된 모션 영역 수보다 크면 모션이 발생되었다고 본다. 알고리즘은 다음과 같다.

```
int three_frame_diff(unsigned char *frame)
{
  초기화;
  for(y=0;y<HEIGHT;y++){
    for(x=0;x<WIDTH;x++){
      이미지 3개를 이용하여 모션이 발생했는지의 유무를 파악;
      if (모션발생){
        모션 발생한 영역의 최소 픽셀의 x,y좌표와 최대 픽셀의 x,y좌표를 확인;
        모션이 발생한 픽셀의 수를 가리키는 변수의 값을 증가;
      }
    }
  }
  if (모션이 발생한 픽셀의 수가 MOTION_AREA보다 크다){
    모션 발생 영역에 사각형을 그린다;
    현재 이미지에 모션이 발생했음을 표시;
  } else {
    현재 이미지에 모션이 발생하지 않았음을 표시;
  }
  /* n은 현재 이미지, n_1은 한 단계 이전 이미지, n_2는 두 단계 이전 이미지를 나타내도록
  업데이트를 한다. */
  n=(n+1)%3;
  n_1=(n_1+1)%3;
  n_2=(n_2+1)%3;
}
```

4.2.2 background_subtraction()

background_subtraction 알고리즘은 배경 이미지를 구성한 뒤 입력되는 이미지의 픽셀값과 배경 이미지의 픽셀값의 차를 이용하여 모션을 검출하는 알고리즘에 대해서 기술한다.

4.2.2.1 자료구조

- 배열
 . unsigned char frame_tmp[WIDTH*HEIGHT];
   현재 캡쳐된 이미지의 y 성분을 저장하기 위한 배열
 . unsigned char frame_bg[WIDTH*HEIGHT];
   배경 이미지의 y 성분을 저장하기 위한 배열
- 변수
 . int init;

초기화가 됐는지를 나타내는 변수(0)

. int minx;

모션이 발생한 영역의 최소 픽셀의 x 좌표를 가리키는 변수

. int miny;

모션이 발생한 영역의 최소 픽셀의 y 좌표를 가리키는 변수

. int maxx;

모션이 발생한 영역의 최대 픽셀의 x 좌표를 가리키는 변수

. int maxy;

모션이 발생한 영역의 최대 픽셀의 y 좌표를 가리키는 변수

. int motion;

한 이미지 내에 모션이 발생한 픽셀의 수를 값으로 가지는 변수

. int moved;

모션이 발생했는지의 유무를 알려주는 변수(0), 0은 모션이 발생하지 않음을 나타냄

4.2.2.2 알고리즘

배경 이미지를 구성하여 현재 캡쳐된 이미지의 픽셀값과 배경 이미지의 픽셀값의 차가 임계값보다 크면 모션이 발생한 픽셀의 수를 가지는 변수의 값을 증가시키고 모션이 발생한 픽셀의 수가 지정된 모션 영역 수보다 크면 모션이 발생되었다고 본다. 알고리즘은 다음과 같다.

```
int background_subtraction(unsigned char *frame)
{
  초기화;
  for(y=0;y<HEIGHT;y++){
    for(x=0;x<WIDTH;x++){
      배경 이미지과 현재 입력된 이미지의 픽셀값의 차를 이용하여 모션의 발생유무를 확인;
      if (픽셀값의 차가 임계값보다 크다){
        moved=1;
        모션 발생한 영역의 최소 픽셀의 x,y좌표와 최대 픽셀의 x,y좌표를 확인;
        모션이 발생한 픽셀의 수를 가리키는 변수의 값을 증가;
      }
      if (모션발생){
        현재 이미지의 픽셀값을 배경 이미지의 픽셀값으로 업데이트;
      } else {
        배경 이미지의 픽셀값과 현재 이미지의 픽셀값에 각각 가중치를 줘서
        배경 이미지의 픽셀값을 업데이트;
      }
    }
  }
```

```
  if (모션이 발생한 픽셀의 수가 MOTION_AREA 보다 크다){
    모션 발생 영역에 사각형을 그린다;
    현재 이미지에 모션이 발생했음을 표시;
  } else {
    현재 이미지에 모션이 발생하지 않았음을 표시;
  }
}
```

4.2.3 hybrid()

hybrid 알고리즘은 카네기 멜론 대학에서 제안한 알고리즘으로 three_frame_diff와 background_subtraction 알고리즘을 접목하여 모션을 검출하는 알고리즘에 대해서 기술한다.

4.2.3.1 자료구조

- 배열

. unsigned char frame_tmp[3][WIDTH*HEIGHT];
  현재 캡쳐된 이미지의 y 성분을 저장하기 위한 배열
. unsigned char frame_bg[3][WIDTH*HEIGHT];
  배경 이미지의 y 성분을 저장하기 위한 배열
. unsigned char ts[3][WIDTH*HEIGHT];
  각 픽셀별로 임계값을 저장하기 위한 배열

- 변수

. int init;
  초기화가 됐는지를 나타내는 변수(0)
. int n;
  현재 이미지를 가리키는 변수(2)
. int n_1;
  한 단계 이전 이미지를 가리키는 변수(1)
. int n_2;
  두 단계 이전 이미지를 가리키는 변수(0)
. int minx;
  모션이 발생한 영역의 최소 픽셀의 x 좌표를 가리키는 변수
. int miny;
  모션이 발생한 영역의 최소 픽셀의 y 좌표를 가리키는 변수
. int maxx;
  모션이 발생한 영역의 최대 픽셀의 x 좌표를 가리키는 변수
. int maxy;

모션이 발생한 영역의 최대 픽셀의 y 좌표를 가리키는 변수

. int motion;

한 이미지 내에 모션이 발생한 픽셀의 수를 값으로 가지는 변수

. int moved;

모션이 발생했는지의 유무를 알려주는 변수(0), 0은 모션이 발생하지 않음을 나타냄

4.2.3.2 알고리즘

하이브리드(hybrid) 알고리즘은 위에서 설명한 three_frame_diff와background_subtraction 알고리즘을 접목한 알고리즘이다. 알고리즘은 다음과 같다.

```
int hybrid(unsigned char *frame)
{
  초기화;
  for(y=0;y=HEIGHT;y++){
    for(x=0;x=WIDTH;x++){
      3 이미지의 픽셀값의 차이를 이용하여 모션이 발생했는지의 유무를 확인;
      if (픽셀값의 차가 임계값보다 크다){
        moved=1;
        모션 발생한 영역의 최소 픽셀의 x,y좌표와 최대픽셀의 x,y좌표를 확인;
        모션이 발생한 픽셀의 수를 가리키는 변수의 값을 증가;
      }
      If(배경 이미지와 현재 이미지의 픽셀값의 차가 임계값보다 크다){
        if(moved==0){
          moved=1;
          모션이 발생한 픽셀의 수를 가리키는 변수의 값을 증가;
        }
      }
      if (모션발생){
        배경값을 현재 이미지의 픽셀값으로 업데이트;
        픽셀당 임계값을 업데이트;
      } else {
        배경 이미지의 픽셀값과 현재 이미지의 픽셀값에 각각 가중치를 줘서
        배경 이미지의 픽셀값을 업데이트;
        픽셀당 임계값을 업데이트;
      }
    }
  }
  if (모션이 발생한 픽셀의 수가 MOTION_AREA 보다 크다){
```

```
    모션 발생 영역에 사각형을 그린다;
    현재 이미지에 모션이 발생했음을 표시;
  } else {
    현재 이미지에 모션이 발생하지 않았음을 표시;
  }
  /* n은 현재 이미지, n_1은 한 단계 이전 이미지, n_2는 두 단계 이전
  이미지를 나타내도록 업데이트를 한다. */
  n=(n+ 1)%3;
  n_1=(n_1+ 1)%3;
  n_2=(n_2+ 1)%3;
}
```

4.3 jpeg 관련 함수

본 절에서는 yuv 포맷 이미지를 jpeg 포맷 이미지로 변환하는 jpeg 관련 함수를 기술한다. jpeg 관련 함수는 jpeg library를 사용한다.

4.3.1 yuv2jpeg()

위에서 말한 바와 같이 yuv 포맷 이미지를 jpeg 포맷 이미지로 압축한다.
함수원형은 int yuv2jpeg(unsigned char *frame2,unsigned char *frame1, int quality)이다.

4.3.1.1 자료구조

- 배열
 . JSAMPROW y[16],cb[16],cr[16];
    이미지의 y, u, v의 각각 값을 가지고 있을 배열
 . JSAMPARRAY raw_array[3];
    이미지의 y, u, v의 주소값을 가지고 있을 배열

- 구조체
 . struct jpeg_compress_struct cjpeg;
    jpeg 이미지의 압축을 위한 정보를 가지고 있는 구조체
 . struct jpeg_error_mgr jerr;
    jpeg 이미지의 에러 관련 정보를 가지고 있는 구조체

- 변수
 . int jpeg_sizes;
    jpeg 파일의 크기를 가진 변수

초기값 0

4.3.1.2 알고리즘

yuv 포맷 이미지를 가진 배열 포인터(frame1)과 jpeg 포맷 이미지 가지고 있을 배열 포인터(frame2)와 jpeg 포맷 이미지의 quality를 입력으로 받으면 이것을 jpeg 포맷 이미지로 변환하여 배열 frame2에 넣고, 변환된 jpeg 포맷 이미지의 크기를 반환한다. 알고리즘은 다음과 같다.

```
int yuv2jpeg(unsigned char *frame2,unsigned char *frame1,int quality)
{
  jpeg 압축을 위한 초기화와 할당;
  압축된 데이타를 위한 목적지(frame2) 상세화;
  압축을 위한 매개변수 설정;
  jpeg 압축 시작;

  이미지를 스캔하고 목적지에 write 한다;

  압축된 jpeg 이미지의 크기값을 jpeg_size에 넣는다;
  jpeg 압축을 끝낸다;
  jpeg 압축을 위해 할당되었던 것을 돌려준다;
  jpeg_size를 반환한다;
}
```

4.4 mpeg 관련 함수

본 절에서는 yuv 포맷 이미지들을 mpeg 포맷 이미지로 변환하여 파일로 만드는 데 필요한 함수에 대해 기술한다. mpeg 함수들은 아래와 같은 mpeg 구조체를 가진다.

```
- mpeg 관련 구조체
struct mpeg{
  AVCodecContext *c;   // AVCodecContext 정보를 가지고 있는 포인터
  AVFrame *picture;    // 이미지 정보를 가지고 있는 포인터
  FILE *f;              // mpeg이 저장된 파일의 정보를 가지는 파일 포인터
  int outbuf_size;     // 파일이 저장될 크기를 가지는 변수
  uint8_t *outbuf;     // 이미지 저장을 위한 포인터
}
```

mpeg 관련 함수들은 ffmpeg library를 사용한다.

4.4.1 mpeg_start()

이 함수는 yuv 포맷 이미지들을 mpeg 포맷으로 변환하여 파일에 저장하기 위해 필요한 설정을 하는 함수이다.
함수원형은 struct mpeg *mpeg_start(unsigned char *image,FILE *f,int width,int height, int bit_rate)이다.

4.4.1.1 자료구조

- 포인터
 . AVCodec *codec;
   AVCodec 정보를 위한 포인터
   초기값 NULL
 . AVCodecContext *c;
   AVCodecContext 정보를 위한 포인터
   초기값 NULL
 . AVFrame *picture;
   AVFrame 정보를 위한 포인터
   초기값 NULL
 . struct mpeg *mpeg;
   mpeg 구조체를 가지는 포인터
 . unsigned char *y, *u, *v;
   yuv 포맷 이미지를 위한 포인터

- 변수
 . int out_size;
   mpeg 파일에 저장하는데 필요한 변수

4.4.1.2 알고리즘

yuv 포맷 이미지를 가지고 있는 배열 포인터(image)와 파일 포인터(f), 이미지의 가로(width) 세로(height) 길이, 그리고 비트율(bit_rate)을 입력으로 받아서 codec을 찾고 AVCodecContext의 관련 정보를 설정하고 codec을 연 후 이미지를 인코딩(encoding)하여 파일에 쓰고, mpeg 구조체를 반환한다. 알고리즘은 다음과 같다.

struct mpeg *mpeg_start(unsigned char *image,FILE *f,int width,int height,int bit_rate)
{
  mpeg 변환을 위한 초기화와 할당;

codec=avcodec_find_encoder(CODEC_ID_MPEG1VIDEO);
AVCodecContext와 AVFrame 할당;
AVCodecContext 관련 정보 설정;
avcodec_open(c,codec);
mpeg->outbuf_size를 큰 크기로 준다(width*height*10);
mpeg->outbuf 할당;
image에서 y, u, v를 나누어 mpeg->picture에 넣는다;
out_size=avcodec_encode_video(mpeg->c,mpeg->outbuf,mpeg->outbuf_size,mpeg->picture);
out_size 만큼 mpeg->f에 mpeg->outbuf를 write 한다;
mpeg을 반환한다;
}

4.4.2 mpeg_put()

이 함수는 mpeg 파일에 yuv 포맷 이미지를 mpeg으로 변환하여 write 하는 것이다.
함수원형은 void mpeg_put(struct mpeg *mpeg,unsigned char *image)이다.

4.4.2.1 자료구조

- 포인터
. unsigned char *y, *u, *v;
yuv 포맷 이미지를 위한 포인터

- 변수
. int out_size;
mpeg 파일에 저장하는데 필요한 변수

4.4.2.2 알고리즘

mpeg 구조체 포인터(mpeg)과 yuv 포맷 이미지를 가지고 있는 배열 포인터(image)를 입력으로 받아서 mpeg->picture에 y, u, v로 나누어 넣은 후 mpeg으로 변환하여 mpeg->f에 write 한다. 알고리즘은 다음과 같다.

void mpeg_put(struct mpeg *mpeg, unsigned char *image)
{
변수 및 배열 포인터 초기화;
image를 y, u, v로 나누어 mpeg->picture에 넣는다;
out_size=avcodec_encode_video(mpeg->c,mpeg->outbuf,mpeg->outbuf_size,mpeg-

```
>picture);
  out_size 만큼 mpeg->f에 mpeg->outbuf를 write 한다;
}
```

4.4.3 mpeg_end()

이 함수는 mpeg 파일을 마무리해주는 것이다.
함수원형은 void mpeg_end(struct mpeg *mpeg)이다.

4.4.3.1 알고리즘

mpeg 구조체 포인터(mpeg)을 입력으로 받아 4byte를 mpeg->f에 write 한 후mpeg 관련 할당을 free 시킨다. 알고리즘은 다음과 같다.

```
void mpeg_end(struct mpeg *mpeg)
{
  mpeg->outbuf[0]=0x00;
  mpeg->outbuf[1]=0x00;
  mpeg->outbuf[2]=0x01;
  mpeg->outbuf[3]=0xb7;
  mpeg->f에 4byte만큼 mpeg->outbuf를 write 한다;
  fclose(mpeg->f);
  free(mpeg->outbuf);
  avcode_close(mpeg->c);
  free(mpeg->c);
  free(mpeg->picture);
  free(mpeg);
}
```

5. 참고문서

[1] ffmpeg, http://ffmpeg.sourceforge.net

[2] Robert T. Collins, Alan J. Lipton, Takeo Kanade, Hironobu Fujiyoshi, David Duggins, Yanghai Tsin, David Tolliver, Nobuyoshi Enomoto, Osamu Hasegawa, Peter Burt and Lambert Wixson, "A System for Video Surveillance and Monitoring: VSAM Final Report," Technical report CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, May, 2000.

[3] video4linux resources, http://www.exploits.org/v4l

제3절 설계 단계 기술 문서

본 절에는 "설계 단계"에서 기술된 문서들을 소개한다. 기술된 문서로는 X 라이브러리와 LessTif을 사용한 디스플레이 방법, ffmpeg을 사용한 mpeg 인코딩 방법, ffmpeg을 사용한 mpeg 디코딩 방법, OSS와 ffmpeg을 사용한 mp2 오디오 레코딩 방법이 있다.

1. X 라이브러리와 LessTif을 사용한 디스플레이 방법

video4linux를 이용하여 실시간으로 화면을 출력하기 위한 X윈도우프로그램 개발을 설명한다. X윈도우 프로그램을 위해 LessTif툴킷을 사용하며 직접 xlib를 사용하기도 한다. video4linux의 사용에 대한 설명은 하지 않는다.

1.1 툴킷

사용자가 요구하는 GUI를 X 라이브러리만 이용하여 프로그래밍 하기에는 너무 어렵고 많은 양의 코딩을 요구한다. 그리고 X 라이브러리만으로 작성한 버튼을 다른 프로그램에서 사용하기 위해서는 프로그램의 소스 코드를 복사하여 사용하여야만 한다. X 윈도우 시스템에서는 사용자가 이러한 객체들을 체계적으로 편리하게 사용할 수 있도록 X 툴킷을 제공한다.
버튼, 메뉴, 스크롤바 등과 같은 캑체들이 미리 작성되어 있다면 사용자는 GUI 작성시 가져다 사용하면 된다. 이렇게 미리 작성된 객체들의 집합을 툴킷이라 한다. 대부분의 툴킷은 X Toolkit Intrinsics(Xt)와 위젯(Widget)으로 구성된다. 이러한 툴킷은 M.I.T의 Athena, OSF의 Motif, AT&T의 OPENLOOK 등 여러 단체에 의해 여러 종류가 개발되었다.

1.2 LessTif

LessTif는 OSF/Motif를 대신하기위한 Hungry Programmers의 버전으로 LGPL(Library Gnu Public License)을 따른다.. OSF/Motif가 오픈 소스이지만 실제적으로 자유소프트의 정의에맞지 않아 실제로 OSF/Motif의 라이센스는 단지 사용하는 것만을 허용하는 정도이다. LessTif는 자유소프트웨어로 OSF/Motif를 대신하기 위한 목적으로 만증어져 Motif로 만들어진 대부분의 프로그램은 LessTif를 통해 수정 없이 사용될 수 있다.

1.3 X 툴킷 프로그램 작성 절차

- 툴킷을 초기화
- 위젯 설정과 생성
- 콜백루틴/이벤트 핸들러가 필요한 경우 등록
- 위젯을 실체화
- 메인 이벤트 처리 순환 시작

1.3.1 툴킷 초기화

툴킷의 초기화는 X 라이브러리에서 클라이언트 프로그램을 작성하기 위해 서버와 연결을 해야 했던것과 같은 역할이라고 볼 수 있다. 툴킷을 초기화 하기 위한 함수로 XtInitialize() 함수가 있다. 이 함수는 서버와 연결하고 최상위 윈도우와 연관된 최상위 위젯을 생성한다. 이 최상위 위젯은 사용자의 클라이언트에서 사용되는 모든 위젯의 최상위 조상 역할을 한다. 이러한 최상위 위젯을 셸(Shell) 위제이라고 부른다.

- Widget XtInitialize(app_name,app_class,options,num_options,argc,argv)
String app_name - 클라이언트의 이름
String app_class - 클라이언트의 클래스 이름
XrmOptionDescRec options[] - 선택 사항들의 리스트
cardinal num_options - 선택 사항들의 개수
cardinal *argc - 명령 라인 매개변수의 개수에 대한 포인터
String argv[] - 처리될 명령 라인 매개변수의 배열

1.3.2 위젯 설정과 생성

사용자는 위젯을 하나의 윈도우와 그 윈도우와 관련된 자료들의 집합으로 생각해도 무방하다. 그러나 사용자는 위젯의 자료구조에 대한 자세한 사항은 몰라도 되지만 각각의 위젯에 설정할 수 있는 매개변수는 알아야 한다.
각 위젯은 계층 구조로 되어 있어 각각의 위젯이 고유의 매개변수를 가지고 있으면 상위계층의 매개변수를 포함하고 있다는 사실에 유의해야 한다.
LessTif의 많은 위젯들의 매개변수를 모두 기억하고 사용하기란 어려우므로 필요한 경우 찾아가면 사용하길 바란다.
LessTif 위젯을 속성과 계층 구조는 LessTif 공심 홈페이지의 Document인 http://www.lesstif.org/Lessdox/lesstif.html 에서 확인할 수 있다.
위젯을 생성하기 위한 방법은 두가지가 있다.
XtSetArg()함수를 사용해서 위젯의 설정을 배열로 만들고 이 배열을 XtCreateManagedWidget()함수를 사용해 생성한다.
또 다른 방법으로는 XtVaCreateManagedWidget()함수를 사용해 직접 설정을 입력해서 만드는 방법이 있다.
샘플코드에서는 XtSetArg()와 XtCreateManagedWidget()함수를 사용하지 않았으므로 예제를 포함한다.

예제 1 - XtCreateManagedWidget() 함수를 사용해 위젯 생성

```
Arg args[20];
Cardinal nargs=0;
Widget button;
...
XtSetArg(args[nargs], XmNwidth, 200); nargs++;  //폭을 200으로 설정
XtSetArg(args[nargs], XmNheight, 200); nargs++; // 높이를 200으로 설정
button = XtCreateManagedWidget("Button", xmPushButtonWidgetClass,
                                    toplevel, (ArgList) args, nargs);
                          // 푸쉬버튼을 toplevel을 부모위젯으로 생성
```

XtVaCreateManagedWidget() 함수의 사용은 XtCreateManagedWidget()함수와거의 같지만 설정을 직접 나열한고 마지막으로 NULL을 주어 마지막임을 알려주면 된다. 사용예는 샘플코드를 참고한다.

1.3.3 콜백루틴/이벤트 핸들러가 필요한 경우 등록

위젯에서 발생되는 이벤트의 처리를 위해, 클라이언트 내에서 콜백함수와 이벤트 핸들러라 불리는 함수 프로그램을 작성할 수 있다. 콜백과 이벤트 핸들러의 차이는 콜백함수가 추상적인 기능을 가지고 있다는 점이다. 예를 들면 한 위젯에서 버튼이 눌려졌을 때의 이벤트의 경우 이벤트 핸들러는 정확하게 대응되는 마스크 ButtonPressMask를 사용하지만 콜백함수는 마우스의 동작을 탐지하는 자원 XmNactivateCallback을 이용하여 함수를 기술할 수 있다.

1.3.3.1 콜백의 등록

각 위젯은 위젯 특정의 조건이 발생할 때 호출될 수 있는 함수를 정의 할 수 있게 한다. 이러한 함수의 리스트를 콜백리스트라 하며 콜백리스트에서 지정한 함수를 콜백함수라 한다. 각 위젯 별로 지원되는 이벤트가 있고 한 위젯에 여러개의 콜백을 추가 할 수 있다. 각 위젯에서 지원되는 콜백은 위에 설명한 LessTif Document에서 확인 할 수 있다.

```
- XtAddCallback(w, callback_name, callback, client_data);
Widget w  -  콜백이 발생할 위젯
String callback_name  -  콜백을 호출하기 위한 동작
XtCallbackProc  -  콜백을 처리할 함수 이름
XtPointer client_data  -  콜백함수에 전해줄 데이터
```

1.3.3.2 이벤트의 등록

이벤트 핸들러란 위젯 내에 특정 유형의 이벤트가 발생할 때 Xt에 의해 호출되는 함수이다. 콜백 등록과 유사한 방법으로 이벤트 핸들러를 등록할 수 있다. Xt는 이벤트 핸들러에 등록된 이벤트 핸들러 루틴을 그 이벤트가 발생하였을 때 실행시킨다.

- XtAddEventHandler(w, event_mask, nonmaskable, proc, client_data);
Widget w - 이벤트가 발생할 위젯
EventMask event_mask - 이벤트 핸들러 루틴을 실행시키기 위한 이벤트
Boolean nonmaskable - 특수한 이벤트, 프로시저 호출,제거 등
XtEventHandler proc - 이벤트 핸들러 루틴 함수
XtPointer client_data - 이벤트 핸들러 루틴에 전달할 데이터

1.3.4 위젯을 실체화

위젯이 생성된 후에는 그 위젯을 화면에 나타내야한다.
XtRealizeWidget()함수를 이용하여 위젯을 실체화 한다.

- XtRealizeWidget(w);
Widget w - 실체화 시킬 최상위 위젯

1.3.5 메인 이벤트 처리 순환 시작

X 라이브러리만을 이용하여 프로그래밍을 하는 경우 프로그래머는 이벤트를 검색하여 그 이벤트가 필요하지 않더라도 프로그램에서 반드시 처리를 해야 한다. 많은 윈도우를 가지는 클라이언트라 가정한다면 각각의 윈도우에서 발생하는 이벤트 처리 루틴이 매우 복잡하게 구성될 것이다. Xt에서는 이벤트의 처리를 하나의 편의함수로 제공하고 있다. 이 함수가 XtMainLoop() 함수이며 이벤트 루프의 역할과 이벤트를 가져오는 역할을 모두 수행한다. 이 함수는 계속해서 이벤트를 처리하며 이 함수가 실행되는 동안 프로그램은 등록된 콜백함수나 이벤트 핸들러에 의한 경우를 제외하고 루프에서 빠져나오지 못한다.

- XtMainLoop();

1.4 X 라이브러리의 사용

많은 X 라이브러리 루틴들은 윈도우와 디스플레이를 직접 다루어 동작한다. 그러나 Xt와 Motif는 윈도우와 디스플레이를 직접 다루지 않고 위젯을 이용하여 사용한다. 물론 위젯은 X 라이브러리를 기반으로 하고 있기 때문에 윈도우를 가지고 있으며 디스플레이에 나타낸다. 그

러므로 Xt는 어떠한 위젯에 대해서도 그들의 값을 나타낼 수 있는 XtWindow()함수와 XtDisplay()함수를 제공한다. 이 함수를 사용해 윈도우와 디스플레이 값을 알아내면 X 라이브러리를 사용해 직접 제어가 가능하게 된다.
샘플코드에서는 영상을 디스플레이 하는데 X 라이브러리의 이미지 처리 함수를 그대로 사용하고 있다.

- Window XtWindow(widget);
- Display *XtDisplay(widget);

1.5 참고문서

[1] LessTif, "LessTif Documentation", http://www.lesstif.org/Lessdox/lesstif.html
[2] XFree86, "Documentation and Resources", http://xfree86.mirror.or.kr/4.4.0
[3] 김충석 역, "X 윈도우시스템 프로그래밍", 이한출판사, 1994.

2. ffmpeg을 사용한 mpeg 인코딩 방법

ffmpeg를 사용하여 video 신호를 mpeg codec로 인코딩하는 방법을 소개한다. 여기서 사용하는 source는 libavcodec의 예제 source인 apiexample.c를 가지고 변형하여 작성한 문서임을 밝힌다. 참고문서가 많지 않아서 정확한 의미를 알지 못하는 부분이 있다. 만약 그러한 부분에 대한 정확한 의미를 아는 사람이 있다면, director@smu.ac.kr로 연락주기 바란다.

2.1 ffmpeg library 사용법

우리가 ffmpeg library를 사용하기 위해서는 동적 라이브러리 파일을 컴파일시 연결시켜 주어야 한다. 동적 라이브러리 파일은 "*.a"파일로서, 우리가 ffmpeg를 통해 인코디하기 위해 사용하는 파일은 "/libavcodec/libavcodec.a"파일이다. 이렇게 ffmpeg library를 사용하는 파일을 컴파일하는 방법은 다음과 같다.

```
-----------------------------------------------------------------------
$ gcc -g -O3 -Wall -o [filename] [filename.c] -lavcodec -lm -lz -ldl
-----------------------------------------------------------------------
```

2.2 ffmpeg 인코딩 방법의 종류

장치가 가지는 신호의 종류는 여러가지이다. 그러나 여기서는 RGB24와YUV420P두가지의 방식을 사용함을 먼저 밝힌다. RGB24와 YUV420P 두가지의 방식을 사용하여 기본적으로 apiexample에서 제공하는 방식과 함수를 사용하는 방식 2가지로 구분하면 총 4가지의 방식이 나오게 된다. 이제부터 이 4가지의 방식에 대해 소개 하려 한다.

2.2.1 인코딩_YUV.c

apiexample.c에서 사용한 기본적인 방식을 사용한 방식이다. AVFrame이라는 구조체를 선헌하고 구조체의 data배열에 실제 영상을 넣어주어 인코딩하는 방식이다.

2.2.2 encoding_YUV_f.c

AVPicture 구조체에 avpicture_fill()함수를 사용하여 영상데이터를 library함수로 넣어주는 방식이다.

2.2.3 encoding_RGB.c

encoding_YUV.c와 방식은 동일하다. context의 세팅값은 PIX_FMT_YUV420P방식으로 세팅되어 있다. 단지 영상을 v4l을 통해서 받아 들일때 우리는 RGB24로 받아오게 되는것이다. RGB24로 받아진 영상은 AVPicture라는 구조체에 저장되고 저장된 AVPicture를

img_convert()를 통해서 YUV420P로 변환한다. 그런후에 AVFrame 구조체의 배열인 data에 넣어주고 인코딩하게 되는것이다.

2.2.4 encoding_RGB_f.c

모든 것은 위의 encoding_RGB.c와 동일하나 단지 apiexample.c에서 영상을 받아오는 방법과 틀릴뿐이다. apiexample.c에서는 영상을 받아와 AVFrame *picture의 data에 넣어 주었다 하지만 여기서는 avpicture_fill()을 사용하여 캡쳐된 영상을 AVPicture 타입의 구조체에 넣어준다. RGB타입인 이 영상을 YUV방식으로 변환하기 위하여 img_convert()함수를 사용하여 RGB24형태의 영상을 YUV420P포맷으로 변환한 후 avcodec_encode_video()를 사용하여 인코드 시키는 것이다.

2.3 v4l에서 영상받기

자세한 내용은 http://pl.smu.ac.kr/researches/projects/linux-dvr/의 2004-02-02-ELF-Video4linux-Capture.txt 문서를 참조 하기 바란다. 여기서는 TV Card와 USB Camera를 세팅하기 위한 부분만 보도록 하겠다.

```
int setChannel(struct video_channel chan, int channel_no, int device_type)
{
  switch(device_type)
  {
    /* Case 1 : TV Card setting */
    case 0 :
      chan.channel = channel_no;
      chan.flags = VIDEO_VC_TUNER;
      chan.type = VIDEO_TYPE_TV;
      chan.norm = VIDEO_MODE_NTSC;
      if((rv = ioctl(video, VIDIOCSCHAN, &chan)) < 0)
      {
        return -1;
      }
      break;
    case 1 :
      chan.channel = channel_no;
      chan.type = VIDEO_TYPE_CAMERA;
      chan.norm = VIDEO_MODE_NTSC;
      if((rv = ioctl(video, VIDIOCSCHAN, &chan)) < 0)
```

```
        {
          return -1;
        }
        break;
      default :
        break;
    }
    return 0;
}
```

2.3.1 TV Card

위의 소스 코드를 보면 TV Card의 경우 device_type의 값이 0일경우 대응됨을 알 수 있다. 이경우 우리는 총 4가지를 설정해준다. channel값과 flags, type, norm값등이다. 이런 세팅값들은 다른 TV Card들에서도 동일하다.
여기서 channel값은 일반적으로 가정용 비디오에서의 Input의 종류를 의미한다. flag값은 비디오 장치가 tuner를 가지고 있다는 것을 알려준다. tuner가 없을 경우 설정하지 않아도 된다. type값은 비디오 장치의 종류를 의미하고 여기서는 TV Card이기 때문에 VIDEO_TYPE_TVnorm값은 NTSC와 PAL이라는 TV신호의 종류를 의미한다.

2.3.2 USB

위의 소스 코드를 보면 USB의 경우 device_type의 값이 1일경우 대응됨을 알 수 있다. 이경우 우리는 3가지를 설정해주는데 channel값과 type, norm값을 설정해 주면된다.

2.4 ffmpeg를 사용하여 인코딩하기

ffmpeg를 사용하여 인코딩을 하기 위해서 우리는 크게 5가지의 단계를 거칠 것이다.
첫번째, avcodec_init();
두번째, avcodec_register_all();
세번째, InitCaptureDevice(DEVICE_NAME, VIDEO_INPUT);
네번째, CaptureImage();
다섯번재, SaveToFile(nMaxCount, map, IMG_W, IMG_H);
첫번째 단계는 avcodec_init()함수를 사용하여 avcodec을 초기화 하는 단계이다.
다음으로 우리는 초기화된 avcodec의 모든 codec을 등록해야 하는데 이일을 avcodec_register_all()함수가 해주게 된다.
그러면 일단 avcodec과 관련된 기본적인 설정은 끝난것이다.
다음으로 우리는 우리가 만든 initCaptureDevice()라는 함수를 부르게 된다. 여기에는

DEVICE_NAME와, VIDEO_INPUT이 인수로 들어가게 되는데 둘다 상수 값으로서 DEVICE_NAME는 TV 또는 USB Camera를 의미하고 VIDEO_INPUT은 말 그대로 input의 종류를 의미한다.
그 다음으로 우리는 설정된 디바이스에서 메모리로 사진을 가져오게 된다. 영상은 여러장의 사진이 모아진 것이므로 우리는 먼저 사진을 찍어야하고 이사진들을 여러장 찍어서 동영상을 만드는 작업을 하게 되는 것이다. 따라서, captureImage()라는 함수를 호출하게 된다. 이 함수는 장치 디바이스로 부터 사진을 받아와 v_map이라는 곳에다 저장하게 된다. 그것을 우리는 in_addr이라는 포인터로 접근 할 수 있다.
이렇게 가져온 사진들을 동영상으로 만들기위해 saveToFile(nMaxCount, map, IMG_W, IMG_H);와 같이 saveToFile()함수를 부르게 된다. 이때 같이 넘겨주는 인수는 총 프레임의 수와 v_map영상이 저장된 곳의 주소(여기서는 in_addr) 포인터, 이미지의 가로와 세로 길이를 넘겨준다. saveToFile()함수는 이러한 인수를 받아서 nMaxCount만큼 돌면서 map주소에서 IMG_W*IMG_H크기 만큼의 사진을 얻어와 우리가 설정해준 값에 따라서 파일로 저장해 준다.
그렇다면 이렇게 저장되기 위해서 우리는 어떠한 설정을 해주어야 하는 것이라는 것을 알수 있다. 우리는 인코딩을 하기 전에 AVContext라는 구조체의 값을 설정해주어야 한다. 이것은 우리가 인코딩할 파일의 정보를 설정하는 것으로서 bitrate, framerate등 영상파일에 대한 정보를 가지고 있다. 자세한 내용은 별첨 문서에서 알아보기로 한다.

2.4.1 saveToFile

```
/*
 *
 * Function      : saveToFile
 *
 * Description : After encoding a frame, save to file
 *
 * Global        : n_frame_count
 *
 * Arguments     : int n_max_count, char *int_addr, int n_width, int n_height
 *
 */
void saveToFile( int n_max_count, char *in_addr, int n_width, int n_height )
{
  unsigned char *y;
  unsigned char *u;
  unsigned char *v;
  int yy;
  int xx;
```

```
int i = 0;

AVCodec *codec = NULL;
AVCodecContext *c = NULL;
int out_size = 0, size = 0, outbuf_size = 0;
AVFrame *picture = NULL;
uint8_t *outbuf = 0, *picture_buf = 0;

if ( n_frame_count == 0 )
{
  f = fopen ( "test.mpg", "w" );
  if ( !f )
  {
    fprintf( stderr, "could not open test.mpg₩n" );
    exit( 1 );
  }
}

if( n_frame_count < n_max_count )
{
  n_frame_count++;
  printf( "Video encoding₩n" );

  /* find the mpeg1 video encoder */
  codec = avcodec_find_encoder( CODEC_ID_MPEG1VIDEO );
  if ( !codec )
  {
    fprintf( stderr, "codec not found₩n" );
    exit( 1 );
  }

  c = avcodec_alloc_context();
  picture = avcodec_alloc_frame();

  /*format*/
  c->get_format = PIX_FMT_YUV420P;
  /* put sample parameters */
  c->bit_rate = 400000;
  /* resolution must be a multiple of two */
```

```
printf("%d, %d₩n", n_width, n_height);
c->width = n_width;
c->height = n_height;
/* frames per second */
c->frame_rate = 30;
c->frame_rate_base= 1;
c->gop_size = 10; /* emit one intra frame every ten frames */

/*maximum number of b frames between non b frames.
  note: the output will be delayed by max_b_frames+ 1 relative to th input
  - encodeing : set by user.
  - decoding: unused*/
c->max_b_frames=1;

/* open it */
if ( avcodec_open( c, codec ) < 0 )
{
  fprintf( stderr, "could not open codec₩n" );
  exit( 1 );
}

/* the codec gives us the frame size, in samples */
/* alloc image and output buffer */
outbuf_size = 100000;
outbuf = malloc( outbuf_size );
size = c->width * c->height;

picture_buf = malloc( ( size * 3 ) / 2 ); /* size for YUV 420 */
picture->data[0] = picture_buf;
picture->data[1] = picture->data[0] +  size;
picture->data[2] = picture->data[1] +  size / 4;
picture->linesize[0] = c->width;
picture->linesize[1] = c->width / 2;
picture->linesize[2] = c->width / 2;

//dp = (unsigned char *)out_addr;
y = ( unsigned char * )in_addr;
u = y +  n_width * n_height;
v = u +  n_width * n_height / 4;
```

```
    fflush( stdout );
    /* prepare a dummy image */
    /* Y */
    for( yy = 0; yy < c->height; yy++ )
    {
      for( xx = 0; xx < c->width; xx++ )
      {
        picture->data[0][yy * picture->linesize[0] + xx] = *y;
        y++;
      }
    }
    /* Cb and Cr */
    for( yy = 0; yy < c->height / 2; yy++ )
    {
      for( xx = 0; xx < c->width / 2; xx++ )
      {
        picture->data[1][yy * picture->linesize[1] + xx] = *u;
        picture->data[2][yy * picture->linesize[2] + xx] = *v;
        u++; v++;
      }
    }

    /* encode the image */
    for( i = 0; i < 2; i++ )
      out_size = avcodec_encode_video( c, outbuf, outbuf_size, picture );

    printf( "encoding frame %3d (size=%5d)₩n", i, out_size );
    fwrite(outbuf, 1, out_size, f);
}

if ( n_frame_count == n_max_count )
{
    n_frame_count++;

    /* get the delayed frames */
    for( ; out_size; i++ )
    {
      fflush( stdout);
      out_size = avcodec_encode_video( c, outbuf, outbuf_size, NULL );
```

```
            printf( "write frame %3d (size=%5d)₩n", i, out_size );
            fwrite( outbuf, 1, out_size, f );
        }

        /* add sequence end code to have a real mpeg file */
        outbuf[0] = 0x00;
        outbuf[1] = 0x00;
        outbuf[2] = 0x01;
        outbuf[3] = 0xb7;
        fwrite( outbuf, 1, 4, f );
        fclose( f );
        free( picture_buf );
        free( outbuf );

        avcodec_close( c );
        free( c );
        free( picture );
        printf( "₩n" );
    }
}
```

위의 소스를 보고 순서 대로 알아보자

여기서는 MPEG1을 사용한 인코딩의 방법만을 알아본다.

우리는 먼저 avcodec_find_encoder() 함수를 사용하여 우리가 사용할 코덱을 찾는다 이 작업이 끝나면 AVContext 형의 c에 avcodec_alloc_context();를 사용하여 메모리 공간을 할당해 준다. 그 다음에 AVFrame 형의 picture에 avcodec_alloc_frame();함수를 사용하여 메모리 공간을 할당하여 준다. 그런후 구조체 c에 값들을 세팅해준다. 여기서 세팅하는 값은 나중에 인코딩을 할경우 인자로 같이 넘겨줄 값이다.

그런 다음 우리는 실제 영상이 들어갈 곳을 실제 영상이 있는 곳과 연결 시켜 준다. 이부분은 RGB이인지 YUV인지에 따라서 틀려지는 부분이 될 것이다. 여기서는 YUV420P포멧이기 때문에 Y, U, V 3개로 나눈다. 이것은 picture의 의 멤버 변수인 data[]배열에 연결 시켜주는데, 우리가 v_map에서 얻어온 in_addr의 주소를 data[0]에 넣어주고 data[1]와 data[2]에는 Y값이 차지하는 공간을 계산하고 in_addr주소에 Y가 차지하는 주소 공간을 더한 값을 넣어주게 된다. 이렇게 세팅을 한후 이미지를 인코딩 하는데 이미지가 처음에는 지연에 의해서 인코딩 되지 않기 때문에 2번의 인코딩을 함으로써 우리는 실제 데이터를 얻을 수 있다. 따라서, for문 안에 avcodec_encode_video(c, outbuf, outbuf_size, picture);를 2번 돌려서 우리가 원하는 압축된 사진을 얻을수 있는 것이다. 이렇게 나온 사진을 우리는 바로 파일에 저장한다. 이렇게 끝난후에 우리는 지연된 이미지가 있는지 알아보기위해 처음부터 out_size만큼 지연된 프레임을 확인하고 다시 파일에 저장 하게 된다. 그런후 파일의 마지막에 mpeg헤더를 추가한후에 모든

자원을 해제하고 끝내게 된다.

2.4.2 포맷에 따는 각각의 MPEG압축 방법

포맷과 함수를 사용하는지 마는지에 따라서 우리는 전부 4가지 경우의 사례를 볼 수 있다. 이러한 부분은 전부가 picture의 data배열에 자료를 넣어주는 방법에서 차이를 보이게 된다. 이러한 차이에 대해서 이제부터 알아볼것이다.

2.4.2.1 YUV420P

```
picture_buf = malloc( ( size * 3 ) / 2 ); /* size for YUV 420 */
picture->data[0] = picture_buf;
picture->data[1] = picture->data[0] + size;
picture->data[2] = picture->data[1] + size / 4;
picture->linesize[0] = c->width;
picture->linesize[1] = c->width / 2;
picture->linesize[2] = c->width / 2;

//dp = (unsigned char *)out_addr;
y = ( unsigned char * )in_addr;
u = y + n_width * n_height;
v = u + n_width * n_height / 4;

fflush( stdout );
/* prepare a dummy image */
/* Y */
for( yy = 0; yy < c->height; yy++ )
{
  for( xx = 0; xx < c->width; xx++ )
  {
    picture->data[0][yy * picture->linesize[0] + xx] = *y;
    y++;
  }
}
/* Cb and Cr */
for( yy = 0; yy < c->height / 2; yy++ )
{
  for( xx = 0; xx < c->width / 2; xx++ )
```

```
        {
          picture->data[1][yy * picture->linesize[1] + xx] = *u;
          picture->data[2][yy * picture->linesize[2] + xx] = *v;
          u++; v++;
        }
      }
```

AVFrame형의 data[0]에는 우리는 원래 YUV420P포맷의 영상을 가지고 있는 주소를 넣어준다. 그런후에 포인터를 Y의 크기 만큼 증가 시킨후에 U의 주소를 data[1]에 넣어주고 V의 주소를 data[2]에 넣어준다. 그런후에 linesize를 정해주는데, Y는 원래 width이고 U,V는 크기가 압축되어 원래 width크기의 반절밖에 안되기 때문에 라인사이즈는 width/2가 된다. 그리고 for문을 돌리면서 각각의 data의 개개별 값에 y, u, v값을 넣어주게 된다.

2.4.2.2 RGB24->YUV420P

```
    picture_buf = malloc((size * 3) / 2); /* size for YUV 420 */
    picture->data[0] = picture_buf;
    picture->data[1] = picture->data[0] + size;
    picture->data[2] = picture->data[1] + size / 4;
    picture->linesize[0] = c->width;
    picture->linesize[1] = c->width / 2;
    picture->linesize[2] = c->width / 2;

    newPicture.data[0] = picture->data[0];
    newPicture.data[1] = picture->data[1];
    newPicture.data[2] = picture->data[2];
    newPicture.linesize[0] = picture->linesize[0];
    newPicture.linesize[1] = picture->linesize[1];
    newPicture.linesize[2] = picture->linesize[2];

    avpicture_fill(&myPicture, in_addr, PIX_FMT_BGR24, c->width, c->height);
    img_convert(&newPicture, PIX_FMT_YUV420P, &myPicture, PIX_FMT_BGR24, c->width, c->height);

    fflush(stdout);
    /* prepare a dummy image */
    /* Y */
    for(y=0;y<c->height;y++)
```

```
    {
      for(x=0;x<c->width;x++)
      {
        picture->data[0][y * picture->linesize[0] + x] = newPicture.data[0][y *
newPicture.linesize[0] + x];
      }
    }

    /* Cb and Cr */
    for(y=0;y<c->height/2;y++)
    {
      for(x=0;x<c->width/2;x++)
      {
        picture->data[1][y * picture->linesize[1] + x] = newPicture.data[1][y *
newPicture.linesize[1] + x];;
        picture->data[2][y * picture->linesize[2] + x] = newPicture.data[2][y *
newPicture.linesize[2] + x];;
      }
    }
```

RGB24->YUV420P와 YUV420P와 다른 점은 인코딩은 YUV420P로 하지만처음에 영상을 받은 후에 이미지를 컨버트 하는 점이다. 우리는 avpicture_fill()이란 함수를 사용해 AVPicture형의 구조체에 영상을 받아오게 된다. 받아온 영상은 RGB24포맷이고 이것을 우리는 img_convert()함수를 사용해 YUV420P포맷으로 변화하게 된다. 변화된 영상은 새로운 AVPicture형의 구조체에 할당 되게 되고 이것을 AVFrame의 구조체의 data[0]부터 data[1], data[2]에 넣어주므로서 그냥 YUV420P포맷을 사용했을때와 별 다를바없이 MPEG 인코딩을 할수 있게 된다.

2.4.2.3 YUV library함수 사용하기

```
    avpicture_fill(picture, in_addr, PIX_FMT_YUV420P, c->width, c->height);
```

위에서 보았던 것과 같이 AVFrame의 data배열에 직접 넣어주어야 했던 실제 데이터 영상을 우리는 avpicture_fill()이라는 함수를 통해서 자동으로 할 수 있다. 하지만, 여기서는 약간의 문제가 생기게 된다. 무엇이냐면, avpicture_fill()함수의 첫번째인자는 즉 데이터가 들어가게 되는 곳의 형이 AVPicture타입이라는 점이다. AVPicture와 AVFrame의 차이점은 별첨 문서에서 확인하게 될것이다. 이런 것은 우리가 원하는 영상을 얻는데 영향을 미치지는 않는다. 하지만, 컴파일시 warning메세지를 출력하게 된다.

이렇게 간단하게 해주고 우리는 인코딩함수를 호출하면 된다.

2.4.2.4 RGB24->YUV420P library함수 사용하기

```
--------------------------------------------------------------------------
avpicture_fill(&temp_picture,buf1,PIX_FMT_BGR24,c->width,c->height);
avpicture_fill(&temp_picture,in_addr,PIX_FMT_BGR24,c->width,c->height);
avpicture_fill(&new_picture,buf,PIX_FMT_YUV420P,c->width,c->height);
img_convert(&new_picture,PIX_FMT_YUV420P,&temp_picture,PIX_FMT_BGR24,c-
        >width,c->height);
picture = (AVFrame *)&new_picture;
--------------------------------------------------------------------------
```

RGB24에서 YUV420P로 변화시키는 과정은 위에서 보았던 함수를 사용하지 않는 것과 차이가 없다. 단지 data배열에 넣는 과정을 함수화 했다는 것이다. 실제 ffmpeg library에 있는 함수를 사용하게 되는것이다. 그리고 형의 충돌을 막기위해 AVFrame형의 picture에 AVPicture형의 new_picture를 넣을때 (AVFrame *)으로 형변환을 해주게 된다.

2.5 참고문서

[1] ffmpeg, http://ffmpeg.sourceforge.net

3. ffmpeg을 사용한 mpeg 디코딩 방법

ffmpeg라이브러리를 사용하여 영상을 Display하는 decoder 프로그램을 기술한다. 3.1에서는 본 프로그램을 수행시키기 위한 환경 설정을 다루며, 3.2에서는 ffmpeg 라이브러리 함수 사용법을 기술한다.

3.1 환경

ffmepg라이브러리를 사용하기 위해서는 우선 ffmpeg을 설치해야 한다. 필자의 개발 환경은 Redhat Linux 9을 설치한 컴퓨터 이며, 본 decoder 프로그램은 비디오 디바이스로는 Samsung의 MPC-C10을 사용하여 필자가 제작한 Encoder프로그램을 대상으로 하고 있다.

기본적으로 시스템은 런레벨 3을 사용하여 Text 기반의 콘솔환경으로 부팅이 되도록 하였다. 환경의 구성은 몇가지 특징만 만족한다면 큰 무리 없이 사용이 가능할 것이다.

3.2 ffmpeg

동영상을 효과적으로 보여주기 위해서 필자가 속한 프로젝트 팀에서는 저작권에서 자유로운 ffmpeg라이브러리를 사용하기로 결정하였다. 본 예제 프로그램도 ffmpeg의 Decoder관련 함수를 사용하여 제작되었음을 알려주는 바이다.

3.2.1 필요한 헤더와 라이브러리 파일

ffmpeg을 설치하였다면 libavcodec디렉토리에 libavcodec.a, avcodec.c, avcodec.h 가 존재함을 알수 있다. avcodec.c와 avcodec.h는 utils.c에 접근한다. utils는 확장 API로써 codec관련 함수들을 정의한다.

3.2.2 초기화

3.2.2.1 코덱 초기화

avcodec_init()을 사용하여 avcodec을 초기화 한다.
avcodec_register_all()은 모든 코덱을 등록하는 함수다.

```
-----------------------------------------------------------------------
avcodec_init();
avcodec_register_all();
-----------------------------------------------------------------------
```

코덱을 사용하기 위해 변수인 AVCodec, AVCodecContext를 정의 한다.

```
-----------------------------------------------------------------------
```

```
AVCodec *codec;
AVCodecContext *c= NULL;
codec = avcodec_find_decoder(CODEC_ID_MPEG1VIDEO);
c= avcodec_alloc_context();

if(codec->capabilities&CODEC_CAP_TRUNCATED)
  c->flags|= CODEC_FLAG_TRUNCATED; // we don't send complete frames
if (!codec)
{
  fprintf(stderr, "codec not found₩n");
  exit(1);
}
```

정의한 AVCodec을 avcodec_open()을 사용하여 AvcodecContext에 등록한다.

```
if (avcodec_open(c, codec) < 0)
{
  fprintf(stderr, "could not open codec₩n");
  exit(1);
}
```

3.2.3 decoding 수행

3.2.3.1 AVFrame 타입으로 디코드

AVFrame은 읽어온 영상 데이타를 저장하는 변수이다.
코덱을 초기화 하였다면, 다음의 함수를 사용하여 영상 데이터를 AVFrame으로 디코딩한다.

```
AVFrame *picture;
uint8_t inbuf[INBUF_SIZE + FF_INPUT_BUFFER_PADDING_SIZE];
int frame, size, got_picture, len;

memset(inbuf + INBUF_SIZE, 0, FF_INPUT_BUFFER_PADDING_SIZE);
..
..
<코덱 초기화>
..
```

```
..
len = avcodec_decode_video(c, picture, &got_picture, inbuf, size);
```

3.2.3.2 AVFrame 데이타 해석

다음은 AVFrame 타입의 영상 데이타를 yuv로 해석한다.

```
int h;
int nSize, nLineSize;
char *pImageBuffer, *pPreImageBuffer;

nSize = nHeight*nWidth;
pImageBuffer = malloc((nSize*3)/2);
pPreImageBuffer = pImageBuffer;

nLineSize = picture->linesize[0];
for(h = 0; h< nHeight; h++)
{
  memcpy(pImageBuffer+ h*nWidth, picture->data[0] + h*nLineSize, nWidth);
}
pImageBuffer += nSize;
nLineSize = picture->linesize[1];

for(h = 0; h< nHeight/2; h++)
{
  memcpy(pImageBuffer+ h*nWidth/2, picture->data[1] + h*nLineSize, nWidth/2);
}

pImageBuffer += nSize/4;
nLineSize = picture->linesize[2];
for(h = 0; h< nHeight/2; h++)
{
  memcpy(pImageBuffer+ h*nWidth/2, picture->data[2] + h*nLineSize, nWidth/2);
}
```

3.2.4 자원해제

사용한 자원은 다음과 같이 해제한다.

```
---------------------------------------------------------------------------
FILE *f;
AVCodec *codec;
AVCodecContext *c= NULL;
AVFrame *picture;
..
..
<코덱초기화>
<AVFrame 타입으로 디코딩>
<AVFrame 데이타 해석>
..
..
fclose(f);
avcodec_close(c);
free(c);
free(picture);
---------------------------------------------------------------------------
```

3.3 참고문서

[1] ffmpeg, http://ffmpeg.sourceforge.net

4. OSS와 ffmpeg을 사용한 mp2 오디오 레코딩 방법

OSS 드라이버를 이용하여 오디오 데이터를 얻고, ffmpeg 라이브러리를 통해 mp2 형식의 압축 포멧으로 저장하는 방법을 설명한다.

4.1 개요

리눅스 DVR에서 사운드와 관련하여 우리가 필요로 하는 기능은 크게 다음과 같이 나눌 수 있다.

- TV 카드의 수신 신호로 부터 사운드 캡쳐하기
- 캡쳐한 사운드를 적당한 포맷으로 압축하여 저장하기
- 압축 저장한 사운드 데이터를 플레이하기

이 문서에서 다루는 부분은 캡쳐한 RAW 포맷의 사운드 소스를 mpeg 1의 레이어 2(다른 말로 mp2라고 함)에 준하는 포맷으로 저장하는 방법에 대한 것이다.

사운드 캡쳐는 기존에 작성한 문서와 소스[1]를 참조하고 mp2 인코딩 부분은 ffmpeg 소스[2]에 있는 apiexample.c 를 참조했다.

4.2 사운드 데이터 저장, 압축을 위한 기초지식

우리가 귀로 듣는 소리는 어떤 물체가 일정한 주기로 떨려 공기로 전파되는 파동이다. 끊어지지 않고 계속 이어지는 파동을 컴퓨터의 데이터로 표현하려면 다음과 같은 개념들이 필요하다.

- Sampling Rate: 1초 동안의 아날로그 신호를 몇개의 샘플(sample)로 표현하였는지 나타내는 수치이다. 단위는 헤르츠(Hz). 음악용 시디에서 사용하는 Sampling Rate는 44100 Hz 이다. 다시 말해 1초 동안의 소리를 44100번의 작은 조각(sample)으로 쪼개어 표현했다는 뜻이다.
- Sample Size: 개개의 샘플 크기를 말한다. 단위는 비트(bit). 음악용 시디에서 쓰는 Sample Size는 보통 16비트이다. 사운드 카드에서 주로 지원하는 샘플 크기도 16비트이다.
- Bit Rate: 이것은 초당 몇 비트의 데이터를 전송할 수 있는지 관한 단위이기 때문에 장치의 성능과 관련이 많다. 단위는 bps(bit per second).

위의 세 가지 요소의 관계는 다음과 같은 식으로 표현된다.

Sampling Rate(Hz) * Sample Size(bit) = Bit Rate(bps)

이해를 돕기 위해 예를 들어보자.
음악시디는 앞에서도 언급했듯이 16비트(bit rate)에 441000Hz(sampling rate)로 아날로그 신

호가 디지틀로 저장된다. 위의 식으로 적용하여 Bit Rate를 구해보면
441000 Hz * 16 Bit = 7056000 bps

7056000 bps는 대략 7 Mbps이다. 여기다 채널의 갯수(스테레오 일때 2, 모노 일때 1)을 곱하면 더 많은 전송률이 필요하게 된다. 이만한 전송률을 네트웍 상에서 가지기는 힘들므로 압축을 하게 되는데, 그 때 압축되어 줄어드는 부분은 바로 이 Bit Rate이다. 다시 말해 압축률의 척도는 Bit Rate인 것이다.

4.3 simple-mp2-encoder의 전체적인 구조

simple-mp2-encoder의 전체적인 구조는 다음과 같다.

- ffmpeg 코덱을 열기 및 초기화
- 압축 시 사용할 메모리 설정
- OSS API로 장치를 열기 및 초기화
- 장치로 부터 받은 사운드 데이터를 압축하고 저장
- 사용한 장치와 메모리 모두 해제

하나씩 구체적으로 알아보자.

4.3.1 ffmpeg 코덱 열기 및 초기화

다음과 같이 초기화 하고 연다.

```
------------------------------------------------------------------------
/* initialize ffmpeg */
avcodec_init();
register_avcodec(&mp2_encoder);

/* find the MP2 encoder */
codec = avcodec_find_encoder(CODEC_ID_MP2);
if (!codec) {
   fprintf(stderr, "codec not foundWn");
   exit(EXIT_FAILURE);
}

c = avcodec_alloc_context();

/* put ffmpeg sample parameters */
```

```
c->bit_rate = BIT_RATE;
c->sample_rate = SAMPLE_RATE;
c->channels = CHANNELS;

/* open codec */
if (avcodec_open(c, codec) < 0) {
  fprintf(stderr, "could not open codec₩n");
exit(1);
```

여기서 쓰이는 함수들은 관용구 같이 써야한다. 다른 방법이 있는지 모르겠지만, 비디오든 오디오든 ffmpeg 라이브러리 사용하여 테이터를 주무를 때 예외없이 나오는 부분들이다. 다만 여기서 주의해서 봐야할 부분은 Sampling Rate, Bit Rate, Channel과 같은 인수값들이다.
mp2와 같은 압축표준은 지정할 수 잇는 인수값들이 정해져 있다.

4.3.2 압축 시 사용할 메모리 설정

한번에 사운드 데이터의 크기와 한번 압축시 저장되는 데이터의 크기를 맞추기 위한 계산이 필요하다.

```
/* the codec gives us the frame size, in samples */
samples_size = c->frame_size * 2 * c->channels;
samples = (short *)malloc(samples_size);
outbuf_size = samples_size * 2;
outbuf = (unsigned char *)malloc(outbuf_size);
```

가장 기초가 되는 값 c->frame_size 이다. 이 값은 ffmpeg이 주어지는 코텍의 종류에 따라 계산하여 알려준다. 이 값을 기초로 샘플의 크기를 계산하고 메모리를 할당한다.
outbuf는 압축한 데이터를 파일에 저장할 때 사용하는 버퍼이다.

4.3.3 OSS API로 장치를 열기 및 초기화

이 부분은 OSS 오디오 프로그래밍[1]에서 잘 설명하였으므로 생략한다. 첨부한 소스에는 가독성을 위해 함수화를 하였다. 중요한 점은 ffmpeg에서 필요한 Bit Rate 대신 Bit Size(AFMT)가 필요하다는 점이다.

4.3.4 장치로 부터 받은 사운드 데이터를 압축하고 저장

초기화가 모두 끝났으므로, 자장할 파일을 열고 장치로 부터 사운드 데이터를 받아 압축하여 저장하는 일만 남았다.

```
/* Open file for encoding */
if ( NULL == (fin = fopen(argv[2], "w")) )
{
  fprintf(stderr, "File %s could not be openedWn", argv[2]);
  exit(EXIT_FAILURE);
}

printf("Start recording sound...Wn");

for (i = 0; i < 500; i++) {
  read(audio, samples, samples_size);
  out_size = avcodec_encode_audio(c, outbuf, outbuf_size, samples);
  fwrite(outbuf, 1, out_size, fin);
}

if ( -1 == ioctl(audio, SNDCTL_DSP_SYNC, 0) )
{
  fprintf(stderr, "ioctl on SNDCTL_DSP_SYNC failedWn %sWn", strerror(errno));
  exit(EXIT_FAILURE);
}
```

중요한 부분은 for 루프인데, 구조는 다음과 같다.
먼저 3.2에서 계산한 샘플 크기만큼 장치로 부터 한번씩 읽어와서 저장한다. 그 다음 저장한 버퍼를 ffmpeg의 avcodec_encode_audio()라는 함수를 사용하여 압축하고, 그것을 미리 메모리 할당한 outbuf에 저장한다. 이 함수의 반환값은 저장된 압축데이터의 크기(byte)이다.
다시 이것을 함수 fwrite()를 사용하여 파일에 저장한다.
이 프로그램에서 임의로 카운트 값(500)을 줬다.

4.3.5 사용한 장치와 메모리 모두 해제

이제 남은 것은 사용한 장치와 메모리를 해제하는 일이다.

```
fclose(fin);
free(outbuf);
```

```
free(samples);
avcodec_close(c);
free(c);
close(audio)
```

4.4 참고문서

[1] Audio File Formats FAG, http://sox.sourceforge.net/AudioFormats.txt

[2] ffmpeg-0.4.8 source, http://pl.smu.ac.kr/lxr/ffmpeg-0.4.8/source

[3] MPEG-2 FAG, http://www.cise.ufl.edu/help/software/doc/mpeg/MPEG-FAQ

[4] 김보람, OSS 오디오 프로그래밍,2004-02-26-KimBoram-OSS-Audio-Programming.txt

제5장 구현 단계

본 장에서는 리눅스 디지틀 비디오 레코더 TV 부분과 Motion 부분 구현에 관해 기술한다. DVR TV와 DVR Motion의 각각 개발 환경과 프로세스별 구현 방법에 대해 기술한다. 마지막으로 구현 단계 기술 문서를 소개한다. 구현 단계 기술 문서로는 ffplay 분석서, video4linux 속도 향상을 위한 double buffering 방법, bt848 및 bt878 칩셋이 장착된 TV 카드 시험, 그리고 세가지 모션 검출 알고리즘이 있다.

제1절 DVR TV 구현

본 절에서는 "리눅스 디지틀 비디오 레코더"의 구현 부분 중 DVR TV를 기술한다.

1. 개발 환경

현재 Linux DVR 소프트웨어 중 TV 부분은 Intel x86 RedHat 9.0(kernel 2.4.20-8, gcc 3.2.2)의 운영체제(OS)를 사용하고 Pentium IV 2.80GHz의 CPU와 512M의 메모리를 사용하고 있다. 또한 캡쳐 디바이스로는 Pinnacle TV, Sigma TV 카드 2가지를 사용하고 있다. 사용하는 소프트웨어로는 OSS(open sound system), ffmpeg 0.4.8을 사용했다. 즉, 현재 사용중인 하드웨어와 소프트웨어는 다음과 같다.

1.1 하드웨어

- CPU: Pentium IV 2.80GHz
- Memory: 512M
- Hard disk: 80G
- TV 카드: Pinnacle TV, Sigma TV

1.2 소프트웨어

- OS: RedHat 9(kernel 2.4.20-8)
- Gcc Compiler: gcc 3.2.2
- OSS: oss-3.8.2
- ffmpeg: ffmpeg-0.4.8

2. 프로세스별 구현

이 절에서는 dvrtv-save, dvrtv-play, dvrtv-recorder, dvrtv-decoder, dvrtv-reservation의 구현을 설명한다. 여기서 이미지는 하나의 정지화상을 의미하고 프레임은 한 이미지가 출력되는 동안의 이미지와 사운드를 의미한다.

2.1 dvrtv-save

dvrtv-save 프로세스는 인수로 TV 장치와 사운드 장치, 그리고 저장할 파일 이름이 주어지면 video4linux와 OSS를 이용하여 수신된 영상 정보를 특별한 형식에 따라 파일에 저장한다.

```
-----------------------------------------------------------------------
void dvrtv-save()
{
(1) video4linux 장치 open 및 설정
(2) OSS 장치 open 및 설정
(3) 설정된 방송채널 번호를 공유메모리에 저장
(4) 공유메모리에 dvrtv-save의 pid 저장
for(;;){
  (5) 사운드 read와 영상 캡쳐
  (6) 사운드와 영상을 버퍼파일에 write
}
-----------------------------------------------------------------------
```

리눅스의 하나의 파일이 가지는 크기의 제한으로 인해 우리는 1개 파일을 2GB까지 만들 수 있다. 따라서, raw포맷으로 구현된 dvrtv-save 프로세스는 여러개의 파일을 연결하여 1개의 버퍼를 만들게 된다. 이렇게 여러개의 파일로 분리된 버퍼를 사용하기 위해 파일을 오픈하거나 닫는 방법이 필요하게 되었다. DVR TV에서는 버퍼를 위해 1개의 파일 디스크립터를 사용하며, 파일을 넘어갈때 마다 전에 사용한 파일을 닫고, 앞으로 사용할 파일을 여는 작업을 한다. 또한, 여러개의 파일을 만들기 위하여 파일이름을 명령행 인수로 받아 "01"에서부터 버퍼파일의 개수까지의 숫자를 붙여 파일을 생성하게 된다. 파일이름과 숫자사이에는 "-"를 사용하여 연결한다.
이러한 작업을 위해 명령행 인수를 통해 받은 파일 이름을 가지고 있을 배열(char file_name[100];)과 숫자를 문자열로 가지고 있을 배열(char number_string[10];)을 사용한다. 그리고, 현재 열려있는 파일의 디스크립터를 로컬변수 int fd로 선언한다.
dvrtv-save에서의 오디오 부분은 TV카드에서 출력된 소리 신호를 사운드카드의 line-in으로 입력받아 디스크에 저장한다. 따라서 필요한 부분은 'v4l에서의 사운드 스트림 출력'과 'OSS를 이용한 사운드 기록'이다.
OSS의 버퍼로 사용할 변수의 크기는 global.h에 선언된 AUDIO_SEC_DATA_SIZE, FRAME_PER_SEC의 값으로 계산한다. 영상 한 프레임에 해당하는 사운드 데이터를 번갈아 가면서 파일에 기록해야하므로 영상 한 프레임에 해당하는 사운드 크기를 버퍼로 할당하는 것이

다.

```
int fd;
char file_name[100];
char number_string[10];
int audio_buffer_size = AUDIO_SEC_DATA_SIZE/FRAME_PER_SEC;
unsigned char audio_buffer[audio_buffer_size];
```

v4l에서의 사운드 스트림 출력은 함수 v4l_set_audio_mute()로 구현되었다. 인수로 TV카드의 파일 기술자와 플래그 값(0:사운드출력,1:사운드출력하지않음)이 들어간다. 코드는 다음과 같다.

(1) video4linux 장치 open 및 설정

```
video_dev=v4l_open(argv[1]);
v4l_set_audio_mute(video_dev,0);
```

OSS를 이용한 사운드 기록은 영상 한 프레임에 해당하는 사운드를 저장하기위한 버퍼를 선언하고, 사운드 카드의 line-in으로 들어온 사운드 소스를 open한 다음, read 와 write를 이용하여 디스크에 저장한다.

(2) OSS 장치 open 및 설정

```
if((audio_dev=open(argv[2],O_RDONLY)) == -1){
   fprintf(stderr,"open: %s₩n",strerror(errno));
   exit(1);
}
oss_init(audio_dev,SAMPLE_RATE,CHANNELS);
```

DVR TV는 두개의 프로세스(dvrtv-save와 dvrtv-play)가 연동하여 하나의 TV로써 역할을 한다. TV 채널변경의 경우 사용자의 입력은 dvrtv-play에서 받지만 실제 변경이 일어나는 프로세스는 dvrtv-save이다. 따라서 두 프로세스 사이에 채널번호가 교환되어야 하는데, 우리는 공유메모리 기술을 이용하였다.
먼저 dvrtv-save 프로세스가 시작할 때, 현재 TV 카드에 설정된 방송 채널번호를 얻어와 공유메모리에 저장한다. 이것은 dvrtv-play가 시작될 때 화면에 현재 방송되는 채널번호를 출력

하는데 사용된다.

---

(3) 설정된 방송 채널번호를 공유메모리에 저장

```
/* get current v4l frequency and save at the shared memory */
station = v4l_get_tuner_station(video_dev);
sprintf(c_station,"%d₩0",station);
shm_station_id=shm_create(STATION_MEM_KEY,8,c_station);
```

그 다음으로 dvrtv-play가 알아야 할 정보는 dvrtv-save의 pid(process id) 이다. 이유는 dvrtv-play에서 사용자가 채널을 변경하는 명령을 내리면 dvrtv-save는 dvrtv-play로 부터 시그널을 받아야 하는데 dvrtv-play가 시그널을 올바로 보내기 위해서는 dvrtv-save의 pid를 알고 있어야 하기 때문이다.

---

(4) 공유메모리에 dvrtv-save의 pid 저장

```
save_pid = getpid();
sprintf(c_save_pid,"%d₩0",save_pid);
shm_pid_id=shm_create(PID_MEM_KEY,10,c_save_pid);
```

영상을 정확히 1초에 정해진 프레임 만큼 저장하기 위해서 DVR TV는 1초를 정해진 프레임으로 나눈후, 나누어진 시간에서 저장을 하게 된다. 만약 저장을 하지 못하고 지연되었을 경우 다음 저장시간에 저장하지 못한 프레임까지 저장을 하게 된다. 이를 위해서 dvrtv-save프로세스에서는 시작시간인 start_time과 현재 시간인 current_time, 시작 시간 부터 현재 시간까지의 시간인 elapse_time을 가지게 된다. 그리고, 지연이 되었는지 안되었는지를 확인하기 위하여 매번 저장한 후에 frame_no를 증가시키고 elapse_time을 초당 프레임 레이트인 TIME_PER_FRAME으로 나눈 후, frame_no를 빼므로서 현재 저장해야할 프레임 수를 n값으로 가지게 된다. 여기서 지연이 되었는지 정상인지는 n값이 1일경우 정상이며, n값이 1보다 클 경우는 지연된 프레임이 있는 경우이다. 만약 n값이 0보다 크지 않으면 현재 저장해야 할 프레임이 없는 경우인데, 이 경우 dvrtv-play는 프레임 레이트에 현재까지의 프레임 숫자에 1을 더해 곱해 다음 프레임까지의 시간을 구한후 현재까지 진행된 시간인 elaspse_time으로 뺀만큼의 시간을 usleep으로 기다리게 한다.
(TIME_PER_FRAME * (frame_no + 1) - elapse_time)

OSS의 read 위치는 영상프레임을 계산하여 저장하는 for 루프 안이고, write는영상 한 프레임

을 디스크에 write한 바로 다음 이어서 버퍼에 저장되어 있던 사운드 소스를 write한다. 이런 방식의 문제점은 영상 프레임이 제 때 캡쳐되지 못하고 밀릴 경우 그 다음 정상적으로 캡쳐된 영상을 밀린 프레임 수만큼 여러번 write하게 되어있는데, 그로 인해 사운드가 정상적으로 캡쳐되더라도 밀린 영상 프레임의 해당하는 사운드 프레임이 저장되지 못하고 여러 번 기록되는 영상 프레임에 해당하는 사운드만 여러번 저장된다는 것이다.
이 문제의 해결책은 사운드를 저장하기 위한 버퍼의 배열로 된 원형큐를 이용하여 영상 캡쳐가 지연되더라도 사운드는 제때 저장되도록 구현하는 것이다.

---

(5) 사운드 read와 영상 캡쳐

```
dvrtv-save(){
  ...
  start_time=get_time();

  for(;;){
    current_time=get_time();

    elapse_time=current_time-start_time;
    n=elapse_time/TIME_PER_FRAME-frame_no;

    if(read(audio_dev,audio_buffer,audio_buffer_size)!=audio_buffer_size){
      fprintf(stderr,"audio read: %s₩n",strerror(errno));
      exit(1);
    }

    if(n>0){
      frame_yuv=v4l_capture(video_dev);
    ...
    } else {
      usleep(TIME_PER_FRAME*(frame_no+ 1)-elapse_time);
    }
  ...
  }
}
```

이미지를 연속해서 저장하기위해 dvrtv-save프로세스는 for문을 돌면서 계속해서 n>0인지를 확인하고 n>0일경우 저장하게 된다. 그러나, 위에서 설명했던것 처럼 파일싸이즈의 제한이 있

기 때문에 연속해서 이미지를 저장하고 있는것으로 모든 것이 끝난 것은 아니다. DVR TV 프로그램에서는 global.h에 버퍼로 사용할 파일의 개수인 SAVE_FILE과 각 파일에 저장할 총시간(단위: 초)인 SAVE_FILE_DURATION과 각 파일의 프레임수인 FRAME_PER_FILE을 정의해두었다. 이렇게 정의 되어있는 각 파일의 시간을 가지고 각각의 버퍼파일의 싸이즈를 계산하여 한 파일의 싸이즈를 넘는 프레임이 저장 된 경우 원래 파일 디스크립터를 닫고, 새로운 파일 디스크립터를 열어 저장하게 된다. 예를 들어, 각 파일에 180개의 프레임이 들어게 된 다면, 180프레임 째가 되면 원래 파일 디스크립터를 닫고 새로운 파일 디스크립터를 열게 된다. 이때 새로운 파일을 열기위해 파일이름 뒤에 붙는 숫자는 현재 프레임을 한 파일에 저장되는 프레임수로 나누고 사용할 버퍼파일의 개수로 모듈러 연산을 하여 나온 값을 취하게 된다.(버퍼파일숫자 = (현재 프레임 수 / FRAME_PER_FILE) % SAVE_FILES)

---

(6) 사운드와 영상을 버퍼파일에 write

```
dvrtv-save(){
  ...
  for(i=0; i<n; i++){
    if(frame_no%(FRAME_PER_FILE)==0){
      if(frame_no%(SAVE_FRAMES*FRAME_PER_FILE)==0){
        strcpy(file_name, argv[3]);
        sprintf(number_string,"-%02d",1);
        strcat(file_name, number_string);
        printf("buffer file start to %s₩n",file_name);
        if((fd = open(file_name,O_RDWR|O_CREAT,0666))==-1){
          fprintf(stderr,"open: %s₩n",strerror(errno));
          exit(1);
        }
        lseek(fd,0,SEEK_SET);
      }
      else {
        close(fd);
        strcpy(file_name, argv[3]);
        sprintf(number_string,"-
%02d",(((frame_no/(FRAME_PER_FILE)))%SAVE_FILES)+ 1);
        strcat(file_name, number_string);
        printf("buffer file jump to %s₩n",file_name);
        if((fd = open(file_name,O_RDWR|O_CREAT,0666))==-1){
          fprintf(stderr,"open: %s₩n",strerror(errno));
          exit(1);
```

```
        }
        lseek(fd,0,SEEK_SET);
      }
    }
    if(write(fd,frame_yuv,WIDTH*HEIGHT*3/2)!=WIDTH*HEIGHT*3/2){
      fprintf(stderr,"video write: %s₩n",strerror(errno));
      exit(1);
    }
    if(write(fd,audio_buffer,audio_buffer_size)!=audio_buffer_size){
      fprintf(stderr,"audio write: %s₩n",strerror(errno));
      exit(1);
    }
    frame_no++;
  }
}
```

---

dvrtv-save의 main() 함수 이외에 dvrtv-play로 부터 시그널을 받을 경우, 그것을 처리하는 시그널 핸들러 함수가 있어야 한다. 이 함수는 시그널이 넘어오면 signal_on변수를 1로 세팅하여 v4l_capture함수에서 채널을 변경 할 수 있게 한다.

```
void catchInt(int signo)
{
  if(signo == SIGUSR1){
    signal_on=1;
  }
}
```

2.2 dvrtv-play

X window에서 DVR TV의 영상을 출력하기 위해 lesstif위젯을 사용해 창을 만든다.
이벤트 입력을 키보드로 처리 하므로 버튼과 같은 위젯은 사용하지 않고 화면을 출력하기 위한 DrawingArea위젯만 사용하고 이벤트를 키 입력으로 처리하기 위해 키처리 함수를 이벤트 핸들러로 등록한다.
dvrtv-play 프로세스에서의 사운드 부분은 저장된 동영상 파일에서 사운드 소스를 읽어 출력하기 때문에 OSS를 사용한다. 사운드의 read와 write는 영상 프레임이 파일로 부터 읽히 직후에 이루어지면 되므로 사운드 장치의 open을 main()에서 할 필요는 없지만 사운드의 볼륨 등

과 같은 부가정보를 화면에 표시하고 조정하기 위해서는 그와 관련된 변수들을 전역으로 설정할 필요가 있다. 이런 이유로 OSS 장치의 open과 초기화 설정은 main()에 넣었다.
dvrtv-save에서와는 반대로 본 프로세스에서는 시작부분에 공유메모리의 내용을 읽어와 전역변수에 그 값들을 저장한다.
먼저 공유메모리에서 dvrtv-save 프로세스가 시작하며 저장한 자신의pid(process id)를 본 프로세스가 읽어와 전역변수 save_pid에 저장한다.
dvrtv-save의 pid를 dvrtv-play 프로세스가 알아야하는 이유는 dvrtv-save 프로세스에게 채널변경시 시그널을 보내야하기 때문이다.

그 다음으로 dvrtv-play 프로세스는 공유메모리로 부터 현재 방송의 채널번호를 알아야 읽어와 전역변수 station에 저장한다. 이 전역변수는 채널변경시나 채널번호의 화면표시 때 사용된다.

```
dvrtv-play(int argc, char **argv)
{
  Widget top_shell, display;

  ...
  if((audio_dev=open("/dev/dsp",O_WRONLY)) == -1){
    fprintf(stderr,"audio open: %s₩n", strerror(errno));
  }
  oss_init(audio_dev,SAMPLE_RATE,CHANNELS);
  volume = oss_mixer_get_volume(audio_dev);

  save_pid_ptr = shm_read(PID_MEM_KEY,10);
  save_pid = atoi(save_pid_ptr);

  station_ptr = shm_read(STATION_MEM_KEY,8);
  station = atoi(station_ptr);

  top_shell = XtInitialize(argv[0], argv[0], NULL, 0, &argc, argv);
  XtAddEventHandler(top_shell, KeyPressMask, FALSE, (XtEventHandler)keyInput,
                    NULL);

  display = XtVaCreateManagedWidget("display", xmDrawingAreaWidgetClass, top_shell,
          XmNwidth, WIDTH, XmNheight, HEIGHT, NULL);
  XtAddEventHandler(top_shell, KeyPressMask, FALSE, (XtEventHandler)keyInput,
                    NULL);
```

```
  XtRealizeWidget(top_shell);
  ...
  XtMainLoop();
}
```

화면 출력은 계속해서 돌아야 하나, X에서는 이벤트를 받기위해 X가 블럭되어 있어야 하므로 화면을 출력하는 부분은 쓰레드로 구현하여 연속해서 화면에 출력하게 했다. 쓰레드가 시작되면 XVideo와 공유메모리를 사용해 출력하기 위한 설정을 한다. 공유메모리를 사용하면 매번 이미지가 바뀔때마다 XCreateImage()를 할 필요 없이 XPutImage()를 할 수 있다. XVideo를 사용해 출력하기 위해 우선 어댑터 ID를 구한다. 그리고 XvShmCreateImag()로 공유메모리로 사용하기 위한 XvImage 변수를 생성하고 이를 공유메모리로 설정한다. 출력할 영상의 포맷은 특별히 포맷 변환 과정 없이 바로 출력할 수 있도록 YUV420P포맷으로 했다.

```
void *play(void *data)
{
  ...
  XvQueryAdaptors(p_dpy, w1, &p_num_adaptors, &adaptor_info);

  for (i = 0; i < p_num_adaptors; i++) {
    xv_port = adaptor_info[i].base_id;
  }

  image = XvShmCreateImage(p_dpy, xv_port, GUID_YUV420_PLANAR, 0,
                           WIDTH, HEIGHT, &yuv_shminfo);
  yuv_shminfo.shmid = shmget(IPC_PRIVATE, image->data_size, IPC_CREAT|0777);
  yuv_shminfo.shmaddr = image->data = shmat(yuv_shminfo.shmid, 0, 0);
  yuv_shminfo.readOnly = False;
  XShmAttach(p_dpy, &yuv_shminfo)

  ...

  for(;;) {
    (1) 타임 쉬프팅
    (2) 파일 이동과 화면 출력
  }
```

타임 쉬프팅을 위해 dvrtv-play는 shift_left_count, shift_right_count, shift_count, buffer_file_num, buffer_full_flag, temp1, temp2, cur 이렇게 3개의 전역 변수와 5개의 로컬 변수를 가진다. 각 로컬 변수에 대해서 간단하게 알아 보자.
shift_left_count는 키처리 함수에서 타임 쉬프팅의 왼쪽 키가 눌렸을 경우 1씩 증가 하게 된다. shift_left_count 변수는 전역변수이다. shift_right_count는 키처리 함수에서 타임 쉬프팅의 왼쪽 키가 한번이라도 눌린 경우 1씩 증가하고 shift_right_count보다 클 수 없다. shift_right_count는 전역 변수이다. shift_count는 live에서 이동되어 있는 카운트 숫자이다. shift_count 전역 변수이다. buffer_file_num은 현재 버퍼파일의 번호이다. buffer_full_flag는 플레이어가 버퍼의 끝을 읽었을 경우 1로 세팅되어 현재 시점에서 뒤로 이동하는데 버퍼의 제일 처음일때 버퍼의 끝으로 이동할 수 있는지 없는 지를 결정한다. buffer_full_flag의 값이 0일 경우는 버퍼의 끝으로 이동하지 못하고, 1일 경우에는 이동할 수 있다.
temp1은 현재 파일에서 뒤로 쉬프팅 할 경우 현재 파일의 위치가 점프하려는 양보다 작을 경우 현재 파일 포인터까지의 양에서 점프할 양에 뺀 값을 가지고 있다. 이 경우 파일 기술자를 전 파일로 바꾼후 파일의 끝에서부터 temp1만큼 뺀 위치로 파일 포인터를 이동시킨다. temp2는 현재 파일에서 앞으로 쉬프팅 할 경우 현재파일 포인터에서 파일의 끝까지의 양이 점프하려는 양보다 작을 경우 버퍼파일 1개의 싸이즈에서 현재 파일포인터까지의 양을 뺀 값을 가지고 있다. 이경우 파일기술자를 다음 파일로 바꾼후 파일의 처음에서 부터 temp2를 더한 위치로 파일 포인터를 이동시킨다. cur은 현재 파일의 파일 포인터의 위치를 나타낸다.

```
int shift_left_count;
int shift_right_count;
int shift_count;
```

```
int buffer_full_flag = 0;
int buffer_file_num = 0;
off_t temp1 = 0;
off_t temp2 = 0;
off_t cur = 0;
```

이미지를 재생하는 부분에 들어가기전 dvrtv-play는 start_time에 현재시간을 넣고 시작하게 된다.

```
start_time=get_time();
```

이미지를 재생하는 부분은 for문 안에서 돌게 된다. for문 안에 처음 들어가게 되면 파일 포인터의 위치를 이동시키기 위해 shift_left_count, shift_right_count가 0보다 큰지 확인한다. 0보다 크지 않을 경우 dvrtv-save와 마찬 가지로 current_time과 elapse_time을 세팅하고 n값을 구하게 된다. n값이 0보다 클경우 실제 영상을 출력하는 부분으로 들어가게 된다. 파일 포인터 역시 1개만 가지고 있고, dvrtv-save와 마찬가지로 파일을 돌아가면서 열게 된다. 단지, 지연이 생겼을 경우 dvrtv-save의 경우 지연이 생긴 프레임을 기억하고 있다가 다음 저장시에 한꺼번에 저장하나 dvrtv-play의 경우는 지연이 생길 경우 지연이 생긴 프레임을 건너뛴다.

파일 포인터를 이동시키는 부분은 다음과 같은 알고리즘으로 움직인다.
키처리 함수에 의해서 shift_left_count나 shift_right_count가 증가하게 되면 각 if문 안으로 들어가게 되는데 이때 현재 파일 포인터의 위치를 얻어온다.

shift_left_count의 경우는 다음과 같다.
shift_left_count가 0보다 클 경우에는 cur이 JUMP*shift_left_count보다 작을 경우 현재 파일을 전 파일로 바꾸기 위한 작업을 하게 된다. 이때, 먼저 temp1에 JUMP*shift_left_count-cur을 설정한다. 그리고, 현재 파일이 제일 첫 파일일 경우에는 buffer_full_flag가 1일 경우에는 파일의 제일 뒤로 이동 해야 하지만 0일 경우에는 이동되지 않고 Starting point of file이라는 메세지를 출력하도록 구현 한다. 반대로 cur이 JUMP*shift_left_count보다 클 경우에는 lseek 함수를 사용해 현재 파일 포인터를 JUMP*shift_left_count만큼 전으로 이동시키게 된다. 마지막으로 shift_left_count를 0으로 설정해준다.
shift_right_count의 경우는 다음과 같다.
shift_right_count가 0보다 클 경우에는 현재 파일 포인터에 버퍼파일 한개의 데이터 사이즈에 현재 파일 포인터를 뺀 값을 설정한다. 그리고, cur의 값이 JUMP*shift_right_count보다 작을 경우 현재 파일을 다음 파일로 바꾸기 위한 작업을 하게 된다. 이때, 먼저 temp에 JUMP*shift_right_count-cur을 설정한다. 그리고, 현재 파일이 제일 마지막 파일일 경우에는 buffer_full_flag의 값을 확인하고, 1일경우에는 파일의 제일 앞으로 이동해야 하지만 0일경우에는 이동 안되고 End point of file이라는 메세지를 출려하도록 구현한다. 반대로 cur의 값이 JUMP*shift_right_count보다 클 경우에는 lseek함수를 사용해 현재 파일 포인터를 JUMP*shift_right_count만큼 후로 이동시키게 된다.

---

(1) 타임 쉬프팅

```
...
if( shift_left_count > 0 ){
   fprintf(stderr, "shift_left_count = %i₩t", shift_left_count);
   if((cur = lseek(fd, 0, SEEK_CUR)) < JUMP*shift_left_count){
      temp1 = JUMP*shift_left_count - cur;
      if(buffer_file_num == 1){
```

```
      if(buffer_full_flag == 0){
        fprintf(stderr, "Starting point of file₩n");
      } else {
        buffer_file_num = SAVE_FILES;

        strcpy(file_name_copy, file_name);
        sprintf(number_string,"-%02d",buffer_file_num);
        strcat(file_name_copy, number_string);
        fprintf(stderr, "%s₩n",file_name_copy);
        if((fd = open(file_name_copy,O_RDONLY))==-1){
          fprintf(stderr,"open: %s₩n",strerror(errno));
          exit(1);
        }
        lseek(fd, -temp1, SEEK_END);
        frame_no = frame_no - (FRAME_PER_SEC*JUMP_TIME*shift_left_count);
        shift_count-=temp1/JUMP+1;
      }
    } else if(buffer_file_num != 1) {
      fprintf(stderr, "buffer_file_num : %i -> ", buffer_file_num);
      buffer_file_num--;
      fprintf(stderr, "%i₩n", buffer_file_num);

      strcpy(file_name_copy, file_name);
      sprintf(number_string,"-%02d",buffer_file_num);
      strcat(file_name_copy, number_string);
      fprintf(stderr, "%s₩n",file_name_copy);
      if((fd = open(file_name_copy,O_RDONLY))==-1){
        fprintf(stderr,"open: %s₩n",strerror(errno));
        exit(1);
      }
      lseek(fd, -temp1, SEEK_END);
      frame_no = frame_no - (FRAME_PER_SEC*JUMP_TIME*shift_left_count);
      shift_count-=temp1/JUMP+1;
    }
  } else {
    lseek(fd, -JUMP*shift_left_count, SEEK_CUR);
    fprintf(stderr, "left shift button clicked₩n");
    frame_no = frame_no - (FRAME_PER_SEC*JUMP_TIME*shift_left_count);
    shift_count-=shift_left_count;
```

```
    }
    shift_left_count=0;
  }
  if( shift_right_count > 0 ){
    fprintf(stderr, "shift_right_count = %i₩t", shift_right_count);
    cur=SAVE_FILE_DURATION*SEC_DATA_SIZE-(lseek(fd, 0, SEEK_CUR));
    if(cur < JUMP*shift_right_count){
      temp2 = JUMP*shift_right_count - cur;
      if(buffer_file_num == SAVE_FILES){
        if(buffer_full_flag == 0){
          fprintf(stderr, "End point of file₩n");
        } else {
          buffer_file_num = 1;
          strcpy(file_name_copy, file_name);
          sprintf(number_string,"-%02d",buffer_file_num);
          strcat(file_name_copy, number_string);
          fprintf(stderr, "%s₩n",file_name_copy);
          if((fd = open(file_name_copy,O_RDONLY))==-1){
            fprintf(stderr,"open: %s₩n",strerror(errno));
            exit(1);
          }
          lseek(fd, temp2, SEEK_SET);
          frame_no = frame_no + (FRAME_PER_SEC*JUMP_TIME*shift_right_count);
          shift_count+=temp1/JUMP+1;
        }
      } else {
        fprintf(stderr, "buffer_file_num : %i -> ", buffer_file_num+1);
        buffer_file_num++;
        fprintf(stderr, "%i₩n", buffer_file_num+1);
        strcpy(file_name_copy, file_name);
        sprintf(number_string,"-%02d",buffer_file_num);
        strcat(file_name_copy, number_string);
        fprintf(stderr, "%s₩n",file_name_copy);
        if((fd = open(file_name_copy,O_RDONLY))==-1){
          fprintf(stderr,"open: %s₩n",strerror(errno));
          exit(1);
        }
        lseek(fd, temp2, SEEK_SET);
        frame_no = frame_no + (FRAME_PER_SEC * JUMP_TIME*shift_right_count);
```

```
        shift_count+ =temp1/JUMP+ 1;
      }
    } else {
      lseek(fd, JUMP*shift_right_count, SEEK_CUR);
      fprintf(stderr, "right shift button clickedWn");
      frame_no = frame_no +  (FRAME_PER_SEC * JUMP_TIME*shift_right_count);
      shift_count+ =shift_right_count;
    }
    shift_right_count = 0;
}
...
```

---

pause_frame_no가 한번 쉬프트 할때 이동하는 프레임 수인 JUMP_TIME*FRAME_PER_SEC과 같으면 shift_count를 감소 시키고, pause_frame_no를 0으로 초기화 시킨다.
elapse_time을 통해 얻어진 n값이 0보다 클 경우 이미지를 출력하는 루틴으로 들어간다. frame_no을 파일당 프레임 수인 FRAME_PER_FILE으로 모듈러 연산을 해서 0일 경우에는 파일의 끝이므로 다음 파일로 넘어간다. 이때 버퍼파일의 처음이나 버퍼파일의 끝일 경우 1번파일을 열어야 하므로 buffer_file_num에 1을 설정한다. 단, 버퍼파일의 끝일 경우는 buffer_full_flag의 값을 1로 설정해야 한다. 파일의 처음이나 끝이 아닌경우는 buffer_file_num을 1씩 증가하고 파일포인터를 닫은후 다음 파일을 연다. 지연이 되어 n값이 1보다 클경우는 지연된 이미지을 다 찍는 것이 아니라 마지막 이미지만을 찍어야 하므로 i=n-1일 경우에만 화면에 출력하는 부분을 부른다.
이때 key_slow의 값이 1이고, key_slow_first_flag가 1이면 슬로우 키가 눌리고 처음 들어온 경우이다. 여기서 쉬프트하는 만큼 전으로 돌리기 위해 shift_left_count를 증가시키고, key_slow_first_flag를 0으로 초기화한다. key_slow가 1일경우 에는 슬로우 키가 눌린 경우이므로 slow_state를 순차적으로 증가 시킨다. slow_state가 증가 되다가 지연시키고 싶은 프레임 수(FRAME_PER_SLOW)로 모듈러 연산한 값이 0이 되면 다음 프레임을 한번 읽어 오게 되고 모듈러 연산이 0이 아닌경우는 전에 모듈러 연산이 0이 되었을 때 읽어온 프레임을 계속 출력하므로서 슬로우 모션이 구현된다.
그리고, 증가 된 slow_state가 슬로우하려는 총 프레임 수인 SLOW_FRAME과 같아지면, key_slow와 slow_state를 0으로 초기화 한다. 이때 shift_count가 0인경우는 현재 시점으로 돌아와야 하는 양이 충분치 않으므로 아무 일도 하지 않고, 0이 아닌경우 shift_right_count를 증가시켜 10초 후로 이동시켜 슬로우한 10초동안 지나간 시간을 되돌린다. 그리고 슬로우를 10초 하지만 10초 동안의 모든 프레임을 보여주지 않으므로 슬로우로 보여주지 않는 나머지 프레임을 lseek()로 이동시킨다.
또한, 일시정지 키가 눌리지 않고, key_slow가 1이고, slow_state를 FRAME_PER_SLOW로 모듈러한 값이 0일 경우와 일시정지 키가 눌리지 않고, key_slow가 눌리지 않았을 경우 영상과 사운드를 읽어온다. 만약 i가 n-1이 아닐 경우 일시정지 키가 눌리지 않았고 슬로우 키가 눌린

상태에서 slow_state를 FRAME_PER_SLOW로 모듈러 연산한 값이 0일 경우 또는 일시정지 키가 눌리지 않았고 슬로우 키가 눌리지 않은 경우는 이미지 싸이즈 만큼을 뛰어 넘고 사운드를 읽어온다.
일시정지 키가 눌리지 않았고 슬로우 키가 눌린 상태에서 slow_state를 5로 모듈러 연산한 값이 0일 경우 또는 일시정지 키가 눌리지 않았고 슬로우 키가 눌리지 않은 경우 사운드를 사운드 버퍼에 쓰고 프레임 숫자를 증가 시킨다.
반대로 일시 정지 키가 눌리거나 슬로우 키가 눌리고 slow_state를 FRAME_PER_SLOW로 모듈러연산한 값이 0이 아닐경우 또는 일시 정지 키가 눌리거나 슬로우 키가 눌리지 않은 경우에는 pause_frame_no를 증가 시킨다. 여기서 pause_frame_no는 앞부분에서 언급 한것처럼 JUMP*FRAME_PER_SEC값과 같아지면 0으로 초기화한다.
마지막으로 n값이 0일 경우에는 TIME_PER_FRAME * (frame_no + 1) - elapse_time 만큼 usleep으로 기다린다.
화면에 이미지를 출력하는 과정은 크게 3부분으로 나뉜다. 우선 이미지에 출력 정보를 그린다. 그리고 창이 확대 또는 축소 된 비율에 따라 창 크기를 조절하고 마지막으로 이미지를 출력한다.
이미지에 정보를 출력하는 것은 항상 하지 않고 info_time에 값이 있을때만 출력한다. info_time에는 앞으로 정보를 출력할 프레임 수가 저장된다. info_time 변수는 키 입력이 있을 때 정해진 프레임 수가 할당되고 화면에 출력될 때마다 하나씩 감소한다. 이미지에 출력 정보는 draw_text()를 사용해 미리 작성된 8x9 크기의 문자를 그린다. 창의 확대 또는 축소 비율은 'zoom = 창의 크기 / 원본 크기'로 비율을 구해서 이미지 출력 크기를 WIDTH*zoom과 HEIGHT*zoom으로 구하고, 또 창의 가로, 세로 비율이 안맞아 창에 여백이 있을경우 가로, 세로 크기를조절한다. 이미지 출력은 공유메모리로 설정된 Image->data에 출력할 내용이 저장된 상태로 XvShmPutImage()를 사용해 X서버가 이미지를 출력하게 한다. 마지막으로 XFlush()로 갱신된 내용을 화면에 뿌리게 한다.
사운드의 read는 앞에서도 언급했듯이 영상 한 프레임이 파일로부터 읽혀지고 파일의 offset이 사운드 데이터의 처음부분에 위치해 있을 때 read하여야 하므로 영상 한 프레임이 read하면 이어서 사운드도 read한다.
그런데 고려해야할 부분은 dvrtv-play가 실행되는 동안 한 프레임의 영상을 어떤 이유에서 읽지 못하게 되면 어차피 화면에 출력하는 시기를 놓치게 되므로 그 영상 프레임을 생략하여 read하지 않는다는 점이다. 영상의 경우 잠깐의 딜레이가 생기더라도 사람은 잘 인식하지 못하는 경향이 있지만 사운드는 그렇지 않고 생략한 만큼 소리가 끊겨나오게 된다. 이런 이유로 영상 한 프레임이 읽혀지지 못하더라도 사운드만큼은 read되어 write되어야 한다. 그래서 영상을 몇 프레임 지나서 읽지못하면 영상 한 프레임 만큼 lseek로 건너뛰는 부분 바로 다음에 사운드는 read를 한다.

---

(2) 파일 이동, 일시 정지, 슬로우모션, 화면 출력

...

if(pause_frame_no == JUMP_TIME*FRAME_PER_SEC) {

```
    shift_count--;
    pause_frame_no = 0;
  }
  current_time=get_time();
  elapse_time=current_time-start_time+(JUMP_TIME*1000*shift_count);
  n=elapse_time/TIME_PER_FRAME-frame_no;
  if( n > 0 ){
    for(i=0; i<n; i++){
      if(frame_no%(FRAME_PER_FILE)==0){
        if((frame_no%SAVE_FRAMES)==0){
          if(frame_no == 0){
            buffer_file_num = 1;
            fprintf(stderr, "%s₩n", file_name);
            strcpy(file_name_copy, file_name);
            sprintf(number_string,"-%02d",buffer_file_num);
            strcat(file_name_copy, number_string);
            fprintf(stderr, "%s₩n",file_name_copy);
            if((fd = open(file_name_copy,O_RDONLY))==-1){
              fprintf(stderr,"open: %s₩n",strerror(errno));
              exit(1);
            }
            lseek(fd,0,SEEK_SET);
          } else {
            buffer_file_num = 1;
            strcpy(file_name_copy, file_name);
            sprintf(number_string,"-%02d",buffer_file_num);
            strcat(file_name_copy, number_string);
            fprintf(stderr, "%s₩n",file_name_copy);
            if((fd = open(file_name_copy,O_RDONLY))==-1){
              fprintf(stderr,"open: %s₩n",strerror(errno));
              exit(1);
            }
            lseek(fd,0,SEEK_SET);
            buffer_full_flag = 1;
          }
        } else {
          fprintf(stderr, "buffer file jump %i ->", buffer_file_num+1);
          buffer_file_num++;
          fprintf(stderr, "%i₩n", buffer_file_num+1);
```

```
      strcpy(file_name_copy, file_name);
      sprintf(number_string,"-%02d",buffer_file_num);
      strcat(file_name_copy, number_string);
      fprintf(stderr, "%s₩n",file_name_copy);
      if((fd = open(file_name_copy,O_RDONLY))==-1){
        fprintf(stderr,"open: %s₩n",strerror(errno));
        exit(1);
      }
      lseek(fd,0,SEEK_SET);
    }
  }
      if(i==(n-1)){
        if(key_slow == 1){
          if(key_slow_first_flag==1){
            shift_left_count++;
            key_slow_first_flag=0;
          }
          slow_state++;
          if(slow_state==SLOW_FRAME){
            if(shift_count!=0){
              shift_right_count++;
              lseek(fd,SLOW_TIME*((FRAME_PER_SEC)-
                    (FRAME_PER_SEC/FRAME_PER_SLOW)),SEEK_CUR);
            }
            key_slow=0;
            slow_state=0;
          }
        }

        if(key_pause!=1 && ((key_slow==1 && (slow_state%5)==0) || key_slow!=1)){
          ret = read(fd, image->data, WIDTH*HEIGHT*3/2);
          read(fd,audio_buffer,audio_buffer_size);
          if(ret != WIDTH*HEIGHT*3/2){
            fprintf(stderr,"read: %s₩n",strerror(errno));
            exit(1);
          }
        }
        /* window's size modify & check zoom */
        XLockDisplay(p_dpy);
```

```
XGetGeometry(p_dpy, w1, &_dw, &_d, &_d, &window_width, &window_height,
                    &_d, &_d);
if(((int)window_height*1.33333333) <= window_width)
  zoom = (((float)window_height)/((float)HEIGHT));
else
  zoom = (((float)window_width)/((float)WIDTH));

XResizeWindow(XtDisplay(top_shell), XtWindow(top_shell),
                    WIDTH*zoom, HEIGHT*zoom);

/* draw frame */
XvShmPutImage(p_dpy, xv_port, w1, gc, image, 0, 0,image->width,
                    image->height, 0, 0,WIDTH*zoom, HEIGHT*zoom, True);

if(info_time>0){
  if(shift_count == 0)
    sprintf(move_string,"    L I V E");
  else
    sprintf(move_string,"   %4d sec",shift_count*JUMP_TIME);
  sprintf(volume_string,"Volume %3d", volume);

  items.font = f_34;
  sprintf(station_ptr,"%d₩0",station);
  items.chars=station_ptr;              // 방송채널번호 화면에 표시
  items.nchars=strlen(items.chars);
  XDrawText(p_dpy, w1, text_gc, WIDTH*zoom-100, 65,&items,1);

  items.font = f_14;
  items.chars=move_string;
  items.nchars=strlen(items.chars);
  XDrawText(p_dpy,w1,text_gc,WIDTH*zoom-190,HEIGHT*zoom-25,&items,1);

  items.font = f_14;
  items.chars=get_date();
  items.nchars=strlen(items.chars);
  XDrawText(p_dpy, w1, text_gc, 30,HEIGHT*zoom-25, &items,1);

  items.font = f_14;
```

```
            items.chars=volume_string;
            items.nchars=strlen(items.chars);
            XDrawText(p_dpy,w1,text_gc,WIDTH*zoom-190,HEIGHT*zoom-55,&items,1);
          }

          XFlush(p_dpy);

          XUnlockDisplay(p_dpy);
//#ifdef DEBUG
          fprintf(stderr,"frame: %d(%d), fps=%4.1f, time: %u, pause: %u₩n",
              frame_no,n,((frame_no+pause_frame_no)/(elapse_time/1000.)),
              current_time, pause_frame_no);
//#endif
        } else {
          if(key_pause!=1 && ((key_slow==1 && (slow_state%FRAME_PER_SLOW)==0)
                || key_slow!=1)){
            lseek(fd,VIDEO_SEC_DATA_SIZE/FRAME_PER_SEC,SEEK_CUR);
            read(fd,audio_buffer,audio_buffer_size);
          }
        }
        if(key_pause!=1 && ((key_slow==1 && (slow_state%FRAME_PER_SLOW)==0) ||
                key_slow!=1)){
          write(audio_dev,audio_buffer,audio_buffer_size);
          frame_no++;
          info_time--;
        }else if(key_pause==1|| ((key_slow==1 && (slow_state%FRAME_PER_SLOW)!=0)
                || key_slow!=1)){
          pause_frame_no++;
        }
      }
    } else {
      fprintf(stderr, "usleep: %i₩n",
          TIME_PER_FRAME*((frame_no+pause_frame_no)+1)-elapse_time);
      usleep(TIME_PER_FRAME*((frame_no+pause_frame_no)+1)
                -elapse_time);
    }
  }
  XFree(image);
  XvStopVideo(XtDisplay(display), xv_port, XtWindow(display));
```

}

...

키 입력이 들어왔을 때는 키 이벤트가 발생해 이벤트 핸들러로 설정된KeyInput()함수가 불려지게 된다. 키 이벤트 발생시 이벤트 핸들러가 불려지면서 넘어오는 변수로부터 눌려진 키와 함께 눌려진 기능키의 상태를 확인해 해당하는 작업을 수행한다. 키보드의 각 키는 각각 고유한 키코드를 가지고 있고 Ctrl, Alt, Shift 등의 기능키가 같이 눌려졌을 때도 각각 고유한 상태번호를 가지고 온다. 이를 switch()를 사용해 구분한다.
'T'키를 누르면 TV 출력을 위해 play 쓰레드를 실행시키고 play 플래그를 1로 바꾼다. 'D'키를 누르면 파일 선택 위젯을 통해 파일을 선택하고 이벤트 핸들러에서 decode 쓰레드와 play 쓰레드를 실행시키고 각각 decode 플래그와 play 플래그를 1로 바꿔 녹화파일 재생을 한다. TV 출력 또는 녹화파일 재생중에 'E'키를 누르면 frame_no, buffer_file_num, shift_count를 0으로 초기화 하고,쓰레드 플래그를 0으로 초기화 해서 쓰레드를 종료해서 대기상태로 나온다.
Shift키와 함께 좌우화살표 키를 누르면 타임 쉬프팅을 위해 shift_left_count 또는 shift_right_count의 값을 증가시킨다.
그 밖에 기능키 없이 'Q'는 종료, 좌우화살표는 채널 증가 또는 감소, 위아래 화살표는 볼륨 증가 또는 감소 기능을 한다. 또 'P'와 'S'는 각각 일시정지와 슬로우모션을 실행시킨다. 볼륨과 관련된 함수 oss_mixer_set_volume()와 oss_mixer_get_volume()는 oss-utils.c에 정의되어 있고 volume은 현재의 볼륨값을 항상 가지고 있는 전역변수이다.

```
XtCallbackProc keyInput(Widget w, XtPointer client, XtPointer call_data)
{
  XKeyPressedEvent *cbs = (XKeyPressedEvent *)call_data;
  pthread_t play_thread;

  switch(cbs->keycode){
    case 24: // 'Q'
      exit(0);
    case 28: // 'T'
      strcpy(file_name, TV_file_name);
      thread_play_flag=1;
      if(pthread_create(&play_thread,NULL,play,NULL)!=0){
        fprintf(stderr,"Thread creation failed");
        exit(1);
      }
      break;
    case 40: // 'D'
```

```
  ...
  file_select_box = XtVaCreateManagedWidget("file_select_box",
                        xmDialogShellWidgetClass, top_shell, XmNselectionLabelString,
                        XmStringCreateLocalized("file_select"), NULL);
  file_select = XtVaCreateManagedWidget("file_select",
          xmFileSelectionBoxWidgetClass, file_select_box, NULL);
  XtUnmanageChild(XtNameToWidget(file_select, "*Help"));

  XtAddCallback(file_select, XmNokCallback, fileSelect_callback, NULL);
  XtAddCallback(file_select, XmNcancelCallback, fileSelect_callback, NULL);
  break;
case 100: // arrow left
  switch((cbs->state)%16){
    ...
    case 1: // with Shift
      if(JUMP_TIME * abs(shift_count) >= SAVE_DURATION){
        fprintf(stderr, "Over shift_left_count₩n");
      } else {
        if(shift_right_count == 0)
          shift_left_count++;
        else shift_right_count--;
      }
      break;
  }
  break;
case 102: // arrow right
  switch((cbs->state)%16){
    ...
    case 1: // with Shift
      ...
      break;
  }
case 98: // arrow up
  ...
      oss_mixer_set_volume(5);
      volume = oss_mixer_get_volume(audio_dev);
  ...
case 104: // arrow down
  ...
```

```
            oss_mixer_set_volume(-5);
            volume = oss_mixer_get_volume(audio_dev);
        ...
    }
}

void fileSelect_callback(Widget w, XtPointer client, XtPointer call_data)
{
    char *fn;
    XmFileSelectionBoxCallbackStruct *cbs;
    cbs = (XmFileSelectionBoxCallbackStruct *) call_data;
    pthread_t play_thread;
    pthread_t decode_thread;

    struct filename_group decode_file;
    {
        char *mpegfilename;
        char *rawfilename;
    };

    if(cbs->reason == XmCR_OK){
        ...
        decode_file.mpegfilename = fn;
        ...
        XtDestroyWidget(XtParent(w));

        decode_file.rawfilename = file_name;

        thread_decode_flag=1;
        if(pthread_create(&decode_thread,NULL,decode,&decode_file)!=0){
            ...
        }
        sleep(3);
        thread_play_flag=1;
        if(pthread_create(&play_thread,NULL,play,NULL)!=0){
            ...
        }
    }
    ...
```

}

---

mpeg 파일을 재생하기 위해서 dvrtv-play에서 decode()쓰레드 함수를 실행시킨다. 디코딩을 수행하기 위해서 먼저 ffmpeg 라이브러리 를 초기화 하여AVFormatContext를 만든다. 만들어진 AVFormatContext를 av_read_packet()의 인수로 넣어 mpeg 파일로 부터 data를 읽어 AVPacket에 저장한다. AVPacket의 stream_index 필드로부터 Audio 또는 Video data인지 구별하여 알맞은 디코딩 과정을 수행한다.

```
decode(void *argv){
  AVFormatContext *ic;
  AVPacket pkt;
  struct filename_group *filename = (struct filename_group *)argv;
  av_open_input_file(&ic, input_filename.mpegfilename, ....);
  av_find_stream_info(ic);

  for(i=0;i<ic->nb_streams;i++){
    enc=&ic->streams[i]->codec;
    codec=avcodec_find_decoder(enc->codec_id);
    avcodec_open(enc, codec);
    switch(enc->codec_type){
      case CODEC_TYPE_AUDIO:
        audio_index=i;
        break;
      case CODEC_TYPE_VIDEO:
        video_index=i;
        break;
    }
  }
  for(;;){
    ret=av_read_packet(ic,pkt);
    if(pkt->stream_index==video_index)
      (3) 이미지 디코딩
    if(pkt->stream_index==audio_index)
      (4) 사운드 디코딩
    av_free_packet(pkt);
  }
}
```

---

decode()함수에서 av_read_packet()을 사용해서 AVPacket->data 필드에 이미지의 데이터를 저장한다. ffmpeg 라이브러리를 사용하여 디코딩한 이미지 데이터 yuv 값은 AVFrame 구조체에 저장된다. AVFrame의 yuv 값을 바이너리 yuv값으로 바꾸기 위해 picture2yuv()함수를 호출한다. 이 데이터를 avcodec_decode_audio()의 인수로 입력하여 이미지 디코딩을 수행한다.

---

(3) 이미지 디코딩

```
AVFrame *picture;
int len=pkt->len, len1;
int ptr=pkt->data;

while(len>0){
  avcodec_decode_video(&ic->streams[index]->codec, picture, &got_picture, ptr, len);
  if(got_picture){
    yuv_frame = picture2yuv(picture);
  }
  ptr+=len1;
  len-=len1;
}
```

---

decode()함수에서 av_read_packet()을 사용해서 AVPacket->data 필드에 사운드의 데이터를 저장한다. 이 데이터를 avcodec_decode_audio()의 인수로 입력하여 사운드 디코딩을 수행한다.

---

(4) 사운드 디코딩

```
int len=pkt->len, len1;
int ptr=pkt->data;

while(len>0){
  avcodec_decode_audio(&ic->streams[index]->codec, audio_buf, &data_size, ptr, len);
  ptr+=len1;
  len-=len1;
}
```

2.3 dvrtv-recorder

이미지 인코딩을 위한 yuv 데이터와 인코딩 데이터는 video_recorder 구조체에저장하며, 다음과 같이 구성된다.

```
---------------------------------------------------------------------
struct video_recorder
{
	AVFrame *picture;
	uint8_t *video_outbuf;
	int video_outbuf_size;
	unsigned char *frame_yuv;
}
---------------------------------------------------------------------
```

사운드 인코딩을 위한 pcm 데이터와 인코딩 데이터는 audio_recorder 구조체에저장하며, 다음과 같이 구성된다.

```
---------------------------------------------------------------------
struct audio_recorder
{
	uint8_t *audio_outbuf;
	int audio_outbuf_size;
	int oss_buffer_size;
	short *oss_buffer;
}
---------------------------------------------------------------------
```

ffmpeg 라이브러리 사용을 위한 초기화 수행후에 반환되는 데이터는mpeg_recorder 구조체에 저장하며, 다음과 같이 구성된다.

```
---------------------------------------------------------------------
struct mpeg_recorder
{
	AVOutputFormat *fmt;
	AVFormatContext *oc;
	AVStream *audio_st;
	AVStream *video_st;
}
---------------------------------------------------------------------
```

이미지를 정확히 1초에 정해진 프레임 만큼 저장하기 위해서 DVR TV는 1초를정해진 프레임으로 나눈후, 나누어진 시간에서 저장을 하게 된다. 만약 저장을 하지 못하고 지연되었을 경우 다음 저장시간에 저장하지 못한 프레임까지 저장을 하게 된다. 이를 위해서 dvrtv-recorder프로세스에서는 시작시간인 start_time과 현재 시간인 current_time, 시작 시간 부터 현재 시간까지의 시간인 elapse_time을 가지게 된다. 그리고, 지연이 되었는지 안되었는지를 확인하기 위하여 매번 저장한 후에 frame_no를 증가시키고 elapse_time을 초당 프레임 레이트인 TIME_PER_FRAME으로 나눈 후, frame_no를 빼므로서 현재 저장해야할 프레임 수를 n값으로 가지게 된다. 여기서 지연이 되었는지 정상인지는 n값이 1일경우 정상이며, n값이 1보다 클 경우는 지연된 프레임이 있는 경우이다. 만약 n값이 0보다 크지 않으면 현재 저장해야 할 프레임이 없는 경우인데, 이 경우 dvrtv-recorder는 프레임 레이트에 현재까지의 프레임 숫자에 1을 더해 곱해 다음 프레임까지의 시간을 구한후 현재까지 진행된 시간인 elaspse_time으로 뺀 만큼의 시간을 usleep으로 기다리게 한다.
(TIME_PER_FRAME * (frame_no + 1) - elapse_time)

```
dvrtv-recorder(){
  ...
  (1) ffmpeg 라이브러리 초기화

  fifo_init(&fifo, size);
  ...
  start_time=get_time();
  for(;;){
    current_time=get_time();
    elapse_time=current_time-start_time;
    n=elapse_time/TIME_PER_FRAME-frame_no;

    if(n>0){
      ret=read(fdraw, video_data.frame_yuv, WIDTH*HEIGHT*3/2);
      ret=read(fdraw, audio_data.oss_buffer, AUDIO_SEC_DATA_SIZE/FRAME_PER_SEC);
      fifo_write(&fifo, (uint8_t *)audio_data.oss_buffer, ret, &fifo.rptr);
      write_video_frame(mpeg, video_data);  <---- (4) 이미지 인코딩
      write_audio_frame(mpeg, audio_data, &fifo); <---- (5) 사운드 인코딩
      frame_no++;
    } else{
      usleep(TIME_PER_FRAME*(frame_no+1)-elapse_time);
    }
  }
  ...
```

}

---

ffmpeg 라이브러리를 사용하기 위해서는 초기화를 수행해야 한다. 우선av_register_all()함수를 호출한다. AVFormatContext의 filename 필드에 mpeg 파일이름을 지정하며, oformat 필드에 AVOutputFormat 데이터를 저장한후 av_set_parameters()함수를 호출하여 AVFormatContext 자료형을 작성한다.
이미지와 사운드 인코딩을 위한 각각 AVStream을 작성한후에 mpeg_recorder구조체에 저장한다.

---

(1) ffmpeg 라이브러리 초기화

```
struct mpeg_recorder mpeg;
const char *filename;

AVOutputFormat *fmt;
AVFormatContext *oc;
AVStream *audio_st;
AVStream *video_st;

av_register_all();
filename = argv[1];
fmt = guess_format(NULL, filename, NULL);
oc = av_mallocz(sizeof(AVFormatContext));
oc->oformat = fmt;
snprintf(oc->filename, sizeof(oc->filename), "%s", filename);
video_st = (2) 이미지 인코딩을 위한 초기화
audio_st = (3) 사운드 인코딩을 위한 초기화
av_set_parameters(oc, NULL)
if(url_fopen(&oc->pb, filename, URL_WRONLY) < 0){
  fprintf(stderr, "Could not open '%s'₩n", filename);
  exit(1);
}
av_write_header(oc);

mpeg.fmt = fmt;
mpeg.oc = oc;
mpeg.audio_st = audio_st;
mpeg.video_st = video_st;
```

return mpeg;

이미지 데이터를 인코딩 하기 위해서는 먼저 AVCodecConetxt, AVStream, AVCodec 자료형을 초기화 해야 한다. AVStream의 codec 필드에 AVCodecContext를 저장한다.
AVCodecContext에는 화질을 지정하는 bit_rate, 초당 프레임을 지정하기 위한 frame_rate, 이미지 가로와 세로 길이인 width, height 등 인코딩 상세정보를 지정한다. codec_id 필드를 인자로 넣어 avcodec_find_encoder(c->codec_id) 함수를 호출한다면 AVCodec을 반환한다. AVCodecContext에 AVCodec 자료형을 등록하기 위해서 avcodec_open()함수를 호출한다.

---

(2) 이미지 인코딩을 위한 초기화

```
AVCodecContext *c;
AVStream *st;
AVCodec *codec;

st = av_new_stream(oc, 0);
c = &st->codec;
c->codec_id = codec_id;
c->codec_type = CODEC_TYPE_VIDEO;

c->bit_rate = 400000;
c->width = WIDTH;
c->height = HEIGHT;
c->frame_rate = FRAME_PER_SEC;
c->frame_rate_base = 1;
c->gop_size = 12;
if(c->codec_id == CODEC_ID_MPEG1VIDEO || c->codec_id == CODEC_ID_MPEG2VIDEO)
{
  c->max_b_frames = 2;
}
codec = avcodec_find_encoder(c->codec_id);
avcodec_open(c,codec)
```

사운드 데이터를 인코딩 하기 위해서는 먼저 AVCodecConetxt, AVStream, AVCodec 자료형을 초기화 해야 한다. AVStream의 codec 필드에 AVCodecContext를 저장한다.
AVCodecContext에는 bit_rate와 sample_rate, channel 필드를 지정하여 음질을 제어한다. codec_id 필드를 인자로 넣어 avcodec_find_encoder(c->codec_id) 함수를 호출한다면

AVCodec을 반환한다. AVCodecContext에 AVCodec 자료형을 등록하기 위해서 avcodec_open()함수를 호출한다.

---

(3) 사운드 인코딩을 위한 초기화

```
AVCodecContext *c;
AVStream *st;
AVCodec *codec;

st = av_new_stream(oc, 1);
c = &st->codec;
c->codec_id = codec_id;
c->codec_type = CODEC_TYPE_AUDIO;
c->bit_rate = BIT_RATE;
c->sample_rate = SAMPLE_RATE;
c->channels = CHANNELS;
codec = avcodec_find_encoder(c->codec_id);
avcodec_open(c, codec)
```

이미지 데이터를 mpeg 파일에 저장하기 위한 절차는 간단하다. 먼저avcodec_encode_video() 함수를 호출하여 yuv 이미지 데이터를 인코딩한다. 인코딩한 데이터를 인자로 넣어 av_write_frame() 함수를 호출하면, mpeg파일에 저장한다.

---

(4) 이미지 인코딩

```
int out_size, ret;
AVCodecContext *c;
AVFrame *picture_ptr;
AVFormatContext *oc = mpeg.oc;
AVStream *st = mpeg.video_st;
AVFrame *picture = video.picture;
int video_outbuf_size = video.video_outbuf_size;
uint8_t *video_outbuf = video.video_outbuf;

c = &st->codec;
mpeg_put_video(picture, c->width, c->height, video.frame_yuv);

picture_ptr = picture;
```

```
out_size = avcodec_encode_video(c, video_outbuf, video_outbuf_size, picture_ptr);
ret = av_write_frame(oc, st->index, video_outbuf, out_size);
```

사운드는 이미지와 달리 연속적인 데이터이다. 그렇기 때문에 AVCodecContext의 frame_rate 필드를 사용할 수 없다.
AVCodecContext에는 frame_size 필드가 있다. 이 필드는 AVCodec에 종속적인 사운드 프레임의 크기이다. av_write_audio()함수를 호출하여 사운드 데이터를 저장할때는 frame_size*channel*2 만큼 저장한다.
dvrtv-save에서 사운드는 초당 25 프레임을 표시하기 위한 크기의 데이터가 저장한다. 이 크기는 frame_size*channel*2 와 다르다. 그렇기 때문에 사운드 데이터를 읽어와 fifo에 저장한다(fifo_write()). fifo에 저장된 사운드 데이터를 frame_size*channel*2 크기만큼 읽어온다면, 사운드 한 프레임 크기만큼의 데이터를 얻을 수 있다(fifo_read()). 얻어온 데이터를 인자로 넣어 avcodec_encode_audio() 함수를 호출하여 사운드 데이터를 인코딩한 후 av_write_frame을 호출하여 mpeg 파일에 저장한다.

---

(5) 사운드 인코딩

```
int out_size;
int frame_bytes;
AVCodecContext *c;
AVFormatContext *oc = mpeg.oc;
AVStream *st = mpeg.audio_st;
FifoBuffer fifo;

fifo_init(&fifo, audio_data.oss_buffer_size);
...
ret=read(fdraw, audio_data.oss_buffer, AUDIO_SEC_DATA_SIZE/FRAME_PER_SEC);
fifo_write(&fifo, (uint8_t *)audio_data.oss_buffer, ret, &fifo.rptr);
...
c = &st->codec;
frame_bytes = c->frame_size*2*c->channels;
while(fifo_read(fifo, (uint8_t *)audio.oss_buffer, frame_bytes, &(fifo->rptr)) == 0){
  out_size = avcodec_encode_audio(c, audio.audio_outbuf,
                                  audio.audio_outbuf_size,audio.oss_buffer);
  av_write_frame(oc, st->index, audio.audio_outbuf, out_size)
}
```

2.4 dvrtv-reservation

dvrtv-reservation 프로세스는 하나의 예약 정보를 입력 받아 예약 시간에 맞춰 DVR TV를 실행시키고 종료시킨다.

```
dvrtv-reservation(int argc, char **argv)
{
  (1) 문자열로 입력받은 예약정보를 구조체에 저장
  (2) 시작시간, 종료시간에 따라 DVR TV실행, 종료 수행
}
```

예약 정보는 프로세스 실행시 시작 년, 월, 일, 현재시간, 분, 종료 년, 월, 일, 시간, 분, 채널, 설명을 'YYYYMMDD/HHMM-YYYYMMDD/HHMM-CH-DESC'의 형식으로 입력한다.

입력받은 예약 정보는 struct reserve 구조체에 저장하며 구조체는 다음과 같이 구성된다.

```
typedef struct reserve {
  struct tm start;
  struct tm end;
  int channel;
  char desc[200];
}
```

문자열로 입력받은 예약 정보를 struct reserve구조체에 저장하기 위해parseTime()함수를 사용한다.

---

(1) 문자열로 입력받은 예약정보를 구조체에 저장

```
char res_info[256];
struct reserve reservation;

strcpy(res_info, argv[1]);
parseTime(&reservation, res_info);
```

---

parseTime()함수에서는 입력받은 문자열에서 년,월,일 등의 각각의 정보를 문자열에서 분리해서 원하는 자료형으로로 바꿔 구조체에 저장한다.

```
---------------------------------------------------------------------
void parseTime(struct reserve *reservation, char *res_info)
{
  strncpy(temp, res_info, 4); temp[4] =  '₩0';
  reservation->start.tm_year = atoi(temp)-1900;
  strncpy(temp, res_info+4, 2); temp[2] =  '₩0';
  reservation->start.tm_mon = atoi(temp)-1;
  strncpy(temp, res_info+6, 2); temp[2] =  '₩0';
  reservation->start.tm_mday = atoi(temp);
  strncpy(temp, res_info+9, 2); temp[2] =  '₩0';
  reservation->start.tm_hour = atoi(temp);
  strncpy(temp, res_info+11, 2); temp[2] =  '₩0';
  reservation->start.tm_min = atoi(temp);

  strncpy(temp, res_info+14, 4); temp[4] =  '₩0';
  reservation->end.tm_year = atoi(temp)-1900;
  strncpy(temp, res_info+18, 2); temp[2] =  '₩0';
  reservation->end.tm_mon = atoi(temp)-1;
  strncpy(temp, res_info+20, 2); temp[2] =  '₩0';
  reservation->end.tm_mday = atoi(temp);
  strncpy(temp, res_info+23, 2); temp[2] =  '₩0';
  reservation->end.tm_hour = atoi(temp);
  strncpy(temp, res_info+25, 2); temp[2] =  '₩0';
  reservation->end.tm_min = atoi(temp);

  strncpy(temp, res_info+28, 2); temp[2] = '₩0';

  reservation->channel = atoi(temp);

  strncpy(reservation->desc, res_info+31, 200);
}
---------------------------------------------------------------------
```

구조체에 저장된 예약 정보로 시작시간이 되면 fork()시스템콜로 새로운 프로세스를 생성하고 exec() 시스템콜을 사용해 DVR TV를 실행시킨다.

---

(2) 시작시간, 종료시간에 따라 DVR TV실행, 종료 수행

```
if((ret = timeCheck(&reservation, START)) == 0) {
  switch(pid = fork()) {
  case 0:
    // DVR TV 실행
    ...
  default:
    if((ret = timeCheck(&reservation, END)) == 0)
      // DVR TV 종료
      ...
  }
}
```

시간을 확인하기 위해 timeCheck()함수를 사용한다.

timeCheck()함수는 매 분마다 현재 시간과 시작시간 또는 종료시간과 일치하는지 확인해서 일치할 경우 0을 반환한다. 시간이 일치하는지의 확인은 년,월,일,시,분을 순서대로 if문을 사용해 비교하고 찾고자 하는 시간보다 현재 시간이 뒤일경우 이미 지나간 것이므로 -1을 반환한다.

예약을 분 단위로 하고 불필요하게 시간을 계속 확인하고 있을 필요가 없으므로 매 분마다 한 번 확인하고 sleep상태로 대기한다. 분이 바뀔 때까지 sleep하기위해 위해 60초(1분)에서 현재 시간의 초를 뺀 시간을 sleep() 함수의 인수로 사용한다.

```
#define START 0
#define END 1

int timeCheck(struct reserve *reservation, int work)
{
  time_t checktime;
  struct tm *now, *target;

  switch(work){
    case START:
      target = &(reservation->start); break;
    case END:
      target = &(reservation->end); break;
  }
```

```
  while(1){
    checktime = time((time_t *) 0);
    now = localtime(&checktime);

    if(now->tm_year == target->tm_year){
      if(now->tm_mon == target->tm_mon){
        if(now->tm_mday == target->tm_mday){
          if(now->tm_hour == target->tm_hour){
            if(now->tm_min == target->tm_min){
              strftime(buffer, 256, "%Y/%m/%d,%H:%M:%S,%a", now);
              printf("time : %s₩n",buffer);

              return(0);
            } else if(now->tm_min > target->tm_min)
              return(-1);
          } else if(now->tm_hour > target->tm_hour)
            return(-1);
        } else if(now->tm_mday > target->tm_mday)
          return(-1);
      } else if(now->tm_mon > target->tm_mon)
        return(-1);
    } else if(now->tm_year > target->tm_year)
      return(-1);
    sleep(60 - now->tm_sec);
  }
}
```

---

3. 참고문서

[1] ffmpeg Multimedia System, http://ffmpeg.sourceforge.net

[2] Motif Programming Manual, http://www.ist.co.uk/motif/download/6A/6A_book.pdf

[3] Motif Reference Manual, http://www.ist.co.uk/motif/download/6B/6B_book.pdf

[4] Open Sound System Programmer's Guide, http://www.opensound.com/pguide/oss.pdf

[5] W.Richard Stevens, "UNIX Network Programing Volume2(Interprocess Communications)", Prentice Hall, 2002.

[6] xfree86 Manual page, http://xfree86.org/4.4.0/manindex3.html

[7] XVideo예제 프로그램, testxv.c, http://bellet.info/XVideo/testxv.c

[8] 김충석, "X 윈도우 시스템 프로그래밍", 이한 출판사, 1994.

제2절 DVR Motion 구현

본 절에서는 "리눅스 디지틀 비디오 레코더"의 구현 부분 중 DVR Motion을 기술한다.

1. 개발 환경

DVR Motion을 개발한 컴퓨터의 사양은 다음과 같다. 운영체제(OS)는 Intel x86 RedHat 9.0(kernel 2.4.20-8, gcc 3.2.2)을 사용하고 Pentium IV 2.80GHz의 CPU와 512M의 메모리를 사용하고 있다. 또한 캡쳐 디바이스로는 Logitech의 QuickCam Pro 4000 2대를 사용하고 있다. 소프트웨어는 ffmpeg 0.4.8과 Apache Web Server 2.0.49를 사용했다. 즉, 현재 사용중인 하드웨어와 소프트웨어는 다음과 같다.

1.1 하드웨어

- CPU: Pentium IV 2.80GHz
- Memory: 512M
- Hard disk: 30G
- Camera: Logitech QuickCam Pro 4000

1.2 소프트웨어

- OS: RedHat 9(kernel 2.4.20-8)
- Gcc Compiler: gcc 3.2.2
- ffmpeg: ffmpeg-0.4.8
- Apache: httpd-2.0.49

2. 프로세스별 구현

dvrmotion-save, dvrmotion-stream, dvrmotion-record, dvrmotion-autorm의 구현 방법에 대해서 기술한다.

2.1 dvrmotion-save

dvrmotion-save 프로세스는 캡쳐 디바이스를 이용하여 이미지를 읽고 yuv 포맷의 raw 데이터로 파일에 저장한다. 이 프로세스는 캡쳐 디바이스명(/dev/video0와 /dev/video1)과 yuv 포맷이 저장될 파일의 이름(save0.yuv나 save1.yuv)과 카메라의 번호(Camera #0와 Camera #1)를 입력 받는다. 즉, /dev/video0 디바이스에서 캡쳐된 이미지를 save0.yuv 파일에 저장하고 카메라 번호는 Camera #0이다. 현재 동시에 2개의 카메라를 사용하기 때문에 DVR Motion

을 수행하면 두 개의 dvrmotion-save 프로세스가 수행된다.

```
-----------------------------------------------------------------------
dvrmotion-save(int argc, char *argv[])
{
          ...
  fd1=v4l_open(argv[1]);
  start_time=get_time();

  for(;;){
    current_time=get_time();
    elapse_time=current_time-start_time;
    n=elapse_time/TIME_PER_FRAME-frame_no;
    if(n>0){
      frame_yuv=v4l_capture(fd1);
      motion=MOTION(frame_yuv);
      draw_text(frame_yuv,3,HEIGHT-3-9,get_date(),VIDEO_PALETTE_YUV420P);
      draw_text(frame_yuv,WIDTH-3,3,argv[3],VIDEO_PALETTE_YUV420P);
      for(i=0;i<n;i++){
        if(frame_no%SAVE_FRAMES==0)
          lseek(fd2,0,SEEK_SET);
        if(write(fd2,frame_yuv,WIDTH*HEIGHT*3/2)!=WIDTH*HEIGHT*3/2){
          fprintf(stderr,"write: %s₩n",strerror(errno));
          exit(1);
        }
      frame_no++;
      }
    } else{
      usleep(TIME_PER_FRAME*(frame_no+1)-elapse_time);
    }
  }
          ...
}
-----------------------------------------------------------------------
```

dvrmotion-save 프로세스는 앞서 기술된 설계서대로 구현되었다.
dvrmotion-save 프로세스는 v4l을 이용하여 이미지를 캡쳐하는데, v4l을 사용할 수 있도록 초기화를 하고 초기화가 끝났을 때, 현재 시간 정보를 저장한다. 즉, 실제로 이미지를 캡쳐하기 직전의 시간을 저장한다. 실제로 이미지를 캡쳐하기 전에 시간 정보를 이용하여 현재 캡쳐해야

할 이미지의 수를 결정한다. 이미지를 새로 캡쳐할 때마다 모션이 발생했는지를 알아본다. 캡쳐된 이미지에 캡쳐된 시간과 카메라 번호 등의 간단한 정보를 쓴다. 이런 과정을 계속 반복하는 중에 정해진 시간보다 빨리 새로운 이미지를 캡쳐하고자할 때는 sleep을 시킨다. 여기에서 yuv 포맷을 저장하는 파일은 queue 처럼 사용되기 때문에 yuv 파일의 끝에 도달하면 파일 포인터를 yuv 파일의 맨 처음으로 옮겨서 yuv 파일의 처음부터 다시 쓴다.

이 프로세스에서 사용된 함수 중에 v4l_open() 함수는 v4l을 사용하기 위해 기본값을 설정하는 함수이다. get_time() 함수는 현재 시간 정보를 millisecond로 반환한다. v4l_capture() 함수는 v4l을 사용하여 실제로 이미지를 읽어오는 역할을 한다. MOTION()은 구현된 모션 검출 알고리즘 중에서 모션 검출에 사용할 알고리즘이 정의되어 있는 매크로이다.
get_date() 함수는 현재 시간 정보를 스트링으로 반환 하는 함수이다. draw_text() 함수는 캡쳐된 이미지에 간단한 캡쳐 정보를 쓸 수 있도록 한다.

2.2 dvrmotion-stream

dvrmotion-stream 프로세스는 yuv 포맷의 raw 데이터로 파일에 저장된 이미지를 읽어서, jpeg 포맷으로 변환하여 스트리밍을 한다. 이 프로세스는 yuv 포맷으로 저장된 파일명(save0.yuv, save1.yuv)과 포트번호(8090, 8091)와 로그파일명(stream0.log, stream1.log)을 입력 받는다. 즉, save0.yuv 파일의 내용을 읽고 8090 포트를 이용하여 스트리밍을 하고 stream0.log 파일에 로그 정보를 저장한다는 의미이다.

```
#define STATUS_OK "HTTP/1.0 200 OK₩n"
#define STATUS_NOT_FOUND "HTTP/1.0 404 Not Found₩n"
#define CONTENT_MULTI "Content-type:
multipart/x-mixed-replace;boundary=BoundaryString₩n₩n"
#define CONTENT_JPEG "Content-type: image/jpeg₩n"
#define BOUNDARY_STRING "--BoundaryString₩n"
#define END_STRING "--BoundaryString--₩n"

dvrmotion-stream(int argc, char *argv[])
{
            ...
  start_time=get_time();
  current_date=get_date();
  fprintf(logfp,"[%s][%d] log starts₩n",current_date,getpid());
  fflush(logfp);

  while(1){
```

```
current_date=get_date();

client_len=sizeof(struct sockaddr_in);
if((sd1=accept(sd0,(struct sockaddr *)&client,&client_len))==-1){
  //        fprintf(logfp,"[%s][%d] accept() failsWn",current_date,getpid());
  //        fflush(logfp);
  continue;
}
if((fork())==0){
  webin=fdopen(sd1,"r");
  webout=fdopen(sd1,"w");

  if(fgets(line,sizeof(line),webin)==NULL){
    fprintf(logfp,"[%s][%d][%s] fgets() failsWn",
              current_date,getpid(),inet_ntoa(client.sin_addr));
    fflush(logfp);
    fclose(webout);
    fclose(webin);
    exit(1);
  }

  if(sscanf(line,"%s %s %s",method,path,version)!=3){
    fprintf(logfp,"[%s][%d][%s] sscanf() failsWn",
              current_date,getpid(),inet_ntoa(client.sin_addr));
    fflush(logfp);
    fclose(webout);
    fclose(webin);
    exit(1);
  }

  fprintf(webout,STATUS_OK);
  fprintf(logfp,"[%s][%d][%s] starts streamingWn",
            current_date,getpid(),inet_ntoa(client.sin_addr));
  fflush(logfp);
  stream_mjpeg(infile);

  fprintf(logfp,"[%s][%d][%s] ends streamingWn",
            current_date,getpid(),inet_ntoa(client.sin_addr));
      ...
```

```
    }
  }
}
```

---

dvrmotion-stream 프로세스는 앞서 기술된 설계서대로 구현되었다.
STATUS_OK, STATUS_NOT_FOUND, CONTENT_MULTI, CONTENT_JPEG, BOUNDARY_STRING, END_STRING 등의 매크로는 http protocol을 참조하면 된다.
dvrmotion-stream 프로세스는 스트리밍을 하기 전에 날짜 정보를 이용하여 로그 파일을 생성한다. 또한 클라이언트가 요청을 하면 jpeg 포맷으로 인코딩된 이미지를 mjpeg 포맷으로 써서(write) 전송한다.
이 프로세스에서 사용된 함수 중에 stream_mjpeg() 함수는 yuv 포맷의 이미지를 jpeg 포맷으로 변환하여 스트리밍을 한다. 이 함수는 매우 중요한 역할을 하는 함수로 구현은 다음과 같다.

```
void stream_mjpeg(const char *infile)
{
          ...
  current_time=get_time();
  frame_no=((current_time-start_time)/TIME_PER_FRAME)+1;
  seek_start=(frame_no%SAVE_FRAMES)*(WIDTH*HEIGHT*3/2);
  lseek(fd,seek_start,SEEK_SET);

  for(;;){
    current_time=get_time();
    elapse_time=current_time-start_time;
    n=elapse_time/TIME_PER_FRAME-frame_no;

    if(n>0){
      for(i=0;i<n;i++){
        if(frame_no%SAVE_FRAMES==0)
          lseek(fd,0,SEEK_SET);
        if(i==(n-1)){
          if(read(fd,frame_yuv,WIDTH*HEIGHT*3/2)!=WIDTH*HEIGHT*3/2){
            fprintf(stderr,"read: %s₩n",strerror(errno));
            exit(1);
          }

          jpeg_size=yuv2jpeg(frame_jpeg,frame_yuv,WIDTH,HEIGHT,JPEG_QUALITY);
```

```
                    ...
            if(fwrite(frame_jpeg,jpeg_size,1,webout)!=1){
              fprintf(stderr,"fwrite: %s₩n",strerror(errno));
              exit(1);
            }

            fprintf(webout,"₩n");
          } else
            lseek(fd,WIDTH*HEIGHT*3/2,SEEK_CUR);
          frame_no++;
        }
      } else {
        usleep(TIME_PER_FRAME*(frame_no+1)-elapse_time);
      }
    }
          ...
}
```

이 함수는 yuv 포맷의 이미지를 jpeg 포맷으로 변환하기 전의 시간 정보를 이용하여 현재 스트리밍을 해야할 이미지의 수를 결정한다. 이미지의 포맷 변환 및 스트리밍을 할 때에도 dvrmotion-save 프로세스가 수행하는 방식과 같이 정해진 시간보다 빨리 새로운 이미지를 변환하고자할 때에는 sleep을 이용하여 초당 스트리밍을 해야할 이미지의 수를 보장한다.
위에서 사용하는 함수 중에 yuv2jpeg() 함수는 yuv 포맷의 이미지를 jpeg 포맷으로 변환시키는 기능을 한다.

2.3 dvrmotion-record

dvrmotion-record 프로세스는 yuv 포맷의 raw 데이터로 파일에 저장된 이미지를 읽어서, mpeg 포맷으로 변환하여 저장한다. 이 프로세스는 yuv 포맷으로 저장된 파일명(save0.yuv와 save1.yuv)과 mpeg 포맷을 저장할 디렉토리(record0와 record1)를 입력받는다. 즉, save0.yuv 파일을 읽어 record0 디렉토리에 mpeg 포맷의 파일을 저장하게 된다.

```
dvrmotion-record(int argc, char *argv[])
{
          ...
  avcodec_init();
  avcodec_register_all();
```

```
start_time=get_time();
for(;;){
  current_time=get_time();
  elapse_time=current_time-start_time;
  n=elapse_time/TIME_PER_FRAME-frame_no;
  if(n>0){
    for(i=0;i<n;i++){
      if(frame_no%SAVE_FRAMES==0){
        lseek(fd,0,SEEK_SET);
      }
      if(i==(n-1)){
        if(read(fd,frame_yuv,WIDTH*HEIGHT*3/2)!=WIDTH*HEIGHT*3/2){
          fprintf(stderr,"read: %s₩n",strerror(errno));
          exit(1);
        }
        if(frame_yuv[0]==MOTION_YES_MAGIC)
          motion_flag=1;
        else
          motion_flag=0;
        if(mpeg_flag==0){
          path=make_filepath(argv[2],0);
          f=fopen(path,"w");
          if(!f){
            fprintf(stderr,"file %s could not be opened₩n",path);
            exit(1);
          }
          mpeg=mpeg_start(frame_yuv,f,WIDTH,HEIGHT,400000);
          mpeg_flag=1;
        }
        if(mpeg_motion_flag==0&&motion_flag==1){
          path=make_filepath(argv[2],motion_flag);
          mf=fopen(path,"w");
          if(!mf){
            fprintf(stderr,"file %s could not be opened₩n",path);
            exit(1);
          }
          mpeg_motion=mpeg_start(frame_yuv,mf,WIDTH,HEIGHT,400000);
          mpeg_motion_flag=1;
        }
```

```
            mpeg_put(mpeg,frame_yuv);
            if(mpeg_motion_flag==1&&motion_flag==1)
              mpeg_put(mpeg_motion,frame_yuv);

            if((frame_no+1)%max_frame_no==0){
              mpeg_end(mpeg);
              mpeg_flag=0;
            }
            if(((frame_no+1)%motion_max_frame_no==0)&&mpeg_motion_flag==1){
              mpeg_end(mpeg_motion);
              mpeg_motion_flag=0;
            }
          } else
            lseek(fd,WIDTH*HEIGHT*3/2,SEEK_CUR);
          frame_no++;
        }
      } else {
        usleep(TIME_PER_FRAME*(frame_no+1)-elapse_time);
      }
    }
          ...
}
```

---

dvrmotion-record 프로세스 역시 앞서 기술된 설계서대로 구현되었다.
dvrmotion-record 프로세스는 ffmpeg 라이브러리에서 제공하는 함수들을 사용하기 위해서 초기화 과정을 수행한다. 초기화 과정이 끝나면 시작 시간을 저장한다. mpeg 포맷으로 저장하기 전에 현재 시간을 저장하고 시간 정보를 이용하여 현재 저장해야할 이미지의 수를 결정한다. 저장할 이미지의 수가 정해지면 yuv 포맷의 이미지를 읽어 mpeg 포맷으로 저장을 하는데 저장할 이미지에 모션의 발생 유무를 확인한다. mpeg 포맷을 저장할 파일이 있는지를 확인하고 파일이 없다면 mpeg 파일을 생성하고 저장을 할 수 있도록 기본값을 설정한다. 모션이 발생한 이미지를 mpeg 포맷으로 저장하고자할 때, 이전에 mpeg 파일이 있는지를 확인하고 없다면 새로 생성하고 저장을 할 수 있도록 기본값을 설정한다. 설정이 끝난 후에는 yuv 포맷을 mpeg 포맷으로 변환하여 저장한다. 현재 저장 읽는 이미지가 저장하고자 했던 최대 이미지의 수와 같으면 mpeg 저장을 끝낸다. 이 때에, mpeg_motion_flag가 설정되어 있으면 모션이 발생한 이미지를 mpeg 포맷으로 변환하여 저장하는 과정을 끝낸다. 여기에서 yuv 포맷을 저장하는 파일은 queue 처럼 사용되기 때문에 yuv 파일의 끝에 도달하면 파일 포인터를 yuv 파일의 맨 처음으로 옮겨서 yuv 파일의 처음부터 다시 읽는다.

dvrmotion-record 프로세스에서 사용된 함수 중 avcodec_init(), avcodec_register_all() 함수는 ffmpeg 라이브러리를 사용하기 위한 초기화를 하는 함수이다. mpeg_start() 함수는 mpeg 파일을 저장하기 위해서 초기화를 하는 함수이다. mpeg_put() 함수는 yuv 포맷의 이미지를 mpeg 포맷의 이미지로 변환하여 mpeg 파일에 쓴다. mpeg_end() 함수는 mpeg 파일의 끝임을 알리기 위해 4바이트를 mpeg 파일에 쓰고, 할당된 자원들을 반환하도록 한다.

2.4 dvrmotion-autorm

dvrmotion-autorm 프로세스는 쉘 스크립트로서, mpeg 포맷의 파일이 저장된 파일 시스템의 용량을 점검한다. 만약 현재 저장된 파일들의 전체 크기가 파일 시스템의 90% 정도이면 가장 오래 전에 저장된 파일을 찾아서 자동으로 지우는 역할을 한다.

```
---------------------------------------------------------------------
while :
do
  sleep 60
  find_oldest_file
  if [ "${CURRENT_PERCENT}" -ge "${MAX_PERCENT}" ]
  then
    rm -f ${OLDEST_FILE}
    rmdir --ignore-fail-on-non-empty ${INSTALL}/record[01]/video/*
    rmdir --ignore-fail-on-non-empty ${INSTALL}/record[01]/motion/*
    echo $myname: rm done: ${OLDEST_FILE}
  else
    echo $myname: no file removed
  fi
done
---------------------------------------------------------------------
```

dvrmotion-autorm 프로세스는 1분마다 디스크의 용량을 점검한다.
find_oldest_file() 함수를 이용하여 가장 오래 전에 생성된 파일을 찾는다. 만약 현재 저장된 파일들의 전체 크기가 파일 시스템의 90% 정도 이면 가장 오래 전에 생성된 파일을 지우게 된다.
find_oldest_file()는 가장 오래전에 저장된 파일을 찾는 함수이다.

3. 참고문서

[1] ffmpeg, http://ffmpeg.sourceforge.net
[2] http 1.0 protocol, http://www.w3.org/Protocols/rfc1945/rfc1945

[3] Robert T. Collins, Alan J. Lipton, Takeo Kanade, Hironobu Fujiyoshi, David Duggins, Yanghai Tsin, David Tolliver, Nobuyoshi Enomoto, Osamu Hasegawa, Peter Burt and Lambert Wixson, A System for Video Surveillance and Monitoring: VSAM Final Report, Technical report CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, May, 2000.

제3절 구현 단계 기술 문서

본 절에서는 "구현 단계"에서 기술된 문서들을 소개한다. 기술된 문서로는 ffplay 분석서, video4linux 속도향상을 위한 double buffering 방법, bt848 및 bt878 칩셋이 장착된 TV 카드 시험, 그리고 세가지 모션 검출 알고리즘이 있다.

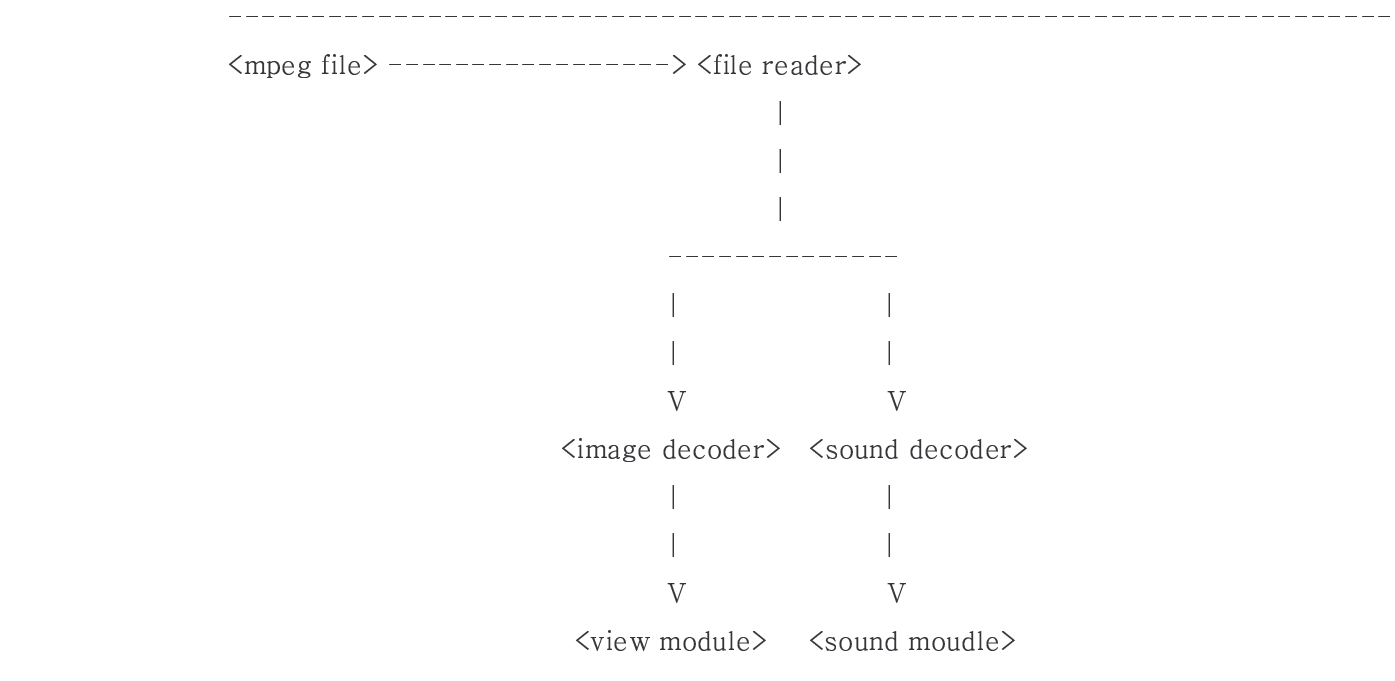
1. ffplay 분석서

본 문서는 ffplay의 동작을 분석한다. 1.1에서는 ffplay의 구조를 기술하며, 1.2에서는 각 구조를 기능별로 설명한다. 1.3에서는 주요 자료구조를 설명한다. 마지막으로 1.4에서는 각 기능에서 호출되는 함수를 분석한다.

1.1 ffplay의 구조도및 기능

Decodempeg의 전체적인 구조를 그림으로 표현하고, 기능을 나열한다.

1.1.1 ffplay의 구조도



1.1.2 ffplay의 기능

- ffplay 초기화
- 동영상정보 읽기

- 동영상정보중 영상정보 decode
- 동영상정보중 소리정보 decode

1.2 기능별 흐름도

ffplay의 각 기능을 기술하며, 동시에 흐름도를 보여준다.

1.2.1 ffplay 초기화

- Register mpegps_demux in AVInputFormat For MPEG

```
------------------------------------------------------------------------
  main()
    |
    V
  av_register_all() -> mpegps_init() -> av_register_input_format()
------------------------------------------------------------------------
```

- Prepare to read a mpeg file

```
------------------------------------------------------------------------
  main()
    |
    V
  stream_open()
    |
    V
  decode_thread()
    |
    V
  av_open_input_file()
    |
    |----------------
    |                |
    V                V
   url_open()      url_fopen()
    |
    V
  url_fdopen()
    |
    V
```

```
  init_put_byte()
-------------------------------------------------------------------------

- Prepare to decode audio and image
-------------------------------------------------------------------------
  main()
    |
    V
  stream_open()
    |
    V
  decode_thread()
    |
    |---------------------------------------
    |                                       |
    V                                       V
  stream_component_open(ViDEO)        stream_component_open(AUDIO)
-------------------------------------------------------------------------
- Prepare to recieve a event
-------------------------------------------------------------------------
  main() ----------> schedule_refresh()
    |
    V
  event_loop()
    |
    V
  SDL_WaitEvent()
-------------------------------------------------------------------------
```

1.2.2 동영상정보 읽기

```
- Read video_data and save in struct PacketQueue
-------------------------------------------------------------------------
  decode_thread() ----------->  av_read_packet()
       |                              |
       |                              V
       |                      mpegps_read_packet()
       |                              |
       |                              V
```

```
        |                                get_byte()
        |                                    |
        |                                    V
        |                               fill_buffer()
        V
  packet_queue_put() : fill_buffer종료후에 호출
```

1.2.3 영상 정보 decode

- Read video_data saved to display

```
  eventloop()
      |
      V
  video_refresh_timer() ------------> schedule_refresh()
      |
      V
  video_display()
      |
      V
  video_image_display() :
```

- decode한 영상정보 저장

```
  video_thread() --------->  packet_queue_get()
     |
     V
  output_picture2()
     |
     V
  queue_picture() :
```

1.3 주요 자료구조

- PacketQueue

```
  {
```

```
    AVPacketList *first_pkt, *last_pkt;
    int nb_packets;
    int size;
    int abort_request;
    SDL_mutex *mutex;
    SDL_cond *cond;
  }
- URLProtocol
  {
    const char *name;
    int (*url_open)(URLContext *h, const char *filename, int flags);
    int (*url_read)(URLContext *h, unsigned char *buf, int size);
    int (*url_write)(URLContext *h, unsigned char *buf, int size);
    offset_t (*url_seek)(URLContext *h, offset_t pos, int whence);
    int (*url_close)(URLContext *h)
    struct URLProtocol *next; /* linked list로 관리한다 */
  }
- URLContext
  {
    struct   URLProtocol prot;
    int flags;
    int is_streamed; /*true if streamed (no seek possible), default =
                       false */
    int max_packet_size; /*if non zero, the stream is packetized with
                           this max packet size*/
    void *priv_data; /*file descriptor를 저장한다*/
    char filename[1]; /*specified filename*/
  }
- ByteIOContext
  {
    unsigned char *buffer; /* file에서 읽어온 data를 저장 */
    int buffer_size;
    unsigned char *buf_ptr, *buf_end;
    void *opaque; /*URLContext를 저장*/
    int (*read_packet)(void *opaque, uint8_t *buf, int buf_size);
    void (*write_packet)(void *opaque, uint8_t, int buf_size);
    int (*seek)(void *opaque, offset_t offset, int whence);
    offset_t pos; /* position in the file of the current buffer */
    int must_flush; /* eof_reached */
```

```
    int eof_reached; /* true if eof reached */
    int write_flag; /* true if open for writing*/
    int is_streamed;
    int max_packet_size;
  }
- AVFormatContext
  {
    /* can only be iformat or oformat, not both at the same time */
    struct AVInputFormat *iformat;
    struct AVOutputFormat *oformat;
    void *priv_data;
    ByteIOContext pb;
    int nb_streams;
    AVStream *streams[MAX_STREAMS];
    char filename[1024]; /* input or output filename */
    /* stream info */
    char title[512];
    char author[512];
    char copyright[512];
    char comment[512];
    char album[512];
    int year;   /* ID3 year, 0 if none */
    int track; /* track number, 0 if none */
    char genre[32]; /* ID3 genre */

    int flags; /* format specific flags */
    /* private data for pts handling (do not modify directly) */
    int pts_wrap_bits; /* number of bits in pts (used for wrapping control) */
    int pts_num, pts_den; /* value to convert to seconds */
    /* This buffer is only needed when packets were already buffered but
       not decoded, for example to get the codec parameters in mpeg streams */
    struct AVPacketList *packet_buffer;

    /* decoding: position of the first frame of the component, in
       AV_TIME_BASE fractional seconds. NEVER set this value directly:
       it is deduced from the AVStream values.   */
    int64_t start_time;
    /* decoding: duration of the stream, in AV_TIME_BASE fractional seconds.
       NEVER set this value directly: it is deduced from the AVStream values.   */
```

```
    int64_t duration;
    /* decoding: total file size. 0 if unknown */
    int64_t file_size;
    /* decoding: total stream bitrate in bit/s, 0 if not
       available. Never set it directly if the file_size and the
       duration are known as ffmpeg can compute it automatically. */
    int bit_rate;
  }
- AVStream
  {
    int index;      /* stream index in AVFormatContext */
    int id;         /* format specific stream id */
    AVCodecContext codec; /* codec context */
    int r_frame_rate;       /* real frame rate of the stream */
    int r_frame_rate_base;/* real frame rate base of the stream */
    void *priv_data;
    /* internal data used in av_find_stream_info() */
    int codec_info_state;
    int codec_info_nb_repeat_frames;
    int codec_info_nb_real_frames;
    /* PTS generation when outputing stream */
    AVFrac pts;
    /* ffmpeg.c private use */
    int stream_copy; /* if TRUE, just copy stream */
    /* quality, as it has been removed from AVCodecContext and put in AVVideoFrame
       MN:dunno if thats the right place, for it */
    float quality;
    /* decoding: position of the first frame of the component, in
       AV_TIME_BASE fractional seconds. */
    int64_t start_time;
    /* decoding: duration of the stream, in AV_TIME_BASE fractional
       seconds. */
    int64_t duration;
  }
- SDL_Overlay
  {
    uint32 format;
    int w, h;
    int planes;
```

```
    uint16 *pitches;
    uint8 **pixels;
    uint32 hw_overlay:1;
  }
- VideoPicture
  {
    double pts; /* presentation time stamp for this picture */
    SDL_Overlay *bmp;
    int width, height; /* source height & width */
    int allocated;
  }
- VideoState
  {
    SDL_Thread *parse_tid;
    SDL_Thread *video_tid;
    AVInputFormat *iformat;
    int no_background;
    int abort_request;
    int paused;
    int last_paused;
    AVFormatContext *ic;
    int dtg_active_format;

    int audio_stream;

    int av_sync_type;
    double external_clock; /* external clock base */
    int64_t external_clock_time;

    double audio_clock;
    double audio_diff_cum; /* used for AV difference average computation */
    double audio_diff_avg_coef;
    double audio_diff_threshold;
    int audio_diff_avg_count;
    AVStream *audio_st;
    PacketQueue audioq;
    int audio_hw_buf_size;
    /* samples output by the codec. we reserve more space for avsync
       compensation */
```

```
    uint8_t audio_buf[(AVCODEC_MAX_AUDIO_FRAME_SIZE * 3) / 2];
    int audio_buf_size; /* in bytes */
    int audio_buf_index; /* in bytes */
    AVPacket audio_pkt;
    uint8_t *audio_pkt_data;
    int audio_pkt_size;
    int64_t audio_pkt_ipts;

    int show_audio; /* if true, display audio samples */
    int16_t sample_array[SAMPLE_ARRAY_SIZE];
    int sample_array_index;
    int last_i_start;

    double frame_timer;
    double frame_last_pts;
    double frame_last_delay;
    double video_clock;
    int video_stream;
    AVStream *video_st;
    PacketQueue videoq;
    int64_t ipts;
    int picture_start; /* true if picture starts */
    double video_last_P_pts; /* pts of the last P picture (needed if B frames are present) */
    double video_current_pts; /* current displayed pts (different from
                                         video_clock if frame fifos are used) */
    int64_t video_current_pts_time; /* time at which we updated video_current_pts - used
                                               to have running video pts */
    VideoPicture pictq[VIDEO_PICTURE_QUEUE_SIZE];
    int pictq_size, pictq_rindex, pictq_windex;
    SDL_mutex *pictq_mutex;
    SDL_cond *pictq_cond;

    //      QETimer *video_timer;
    char filename[1024];
    int width, height, xleft, ytop;
}- PacketQueue
{
    AVPacketList *first_pkt, *last_pkt;
    int nb_packets;
```

```
    int size;
    int abort_request;
    SDL_mutex *mutex;
    SDL_cond *cond;
  }
- URLProtocol
  {
    const char *name;
    int (*url_open)(URLContext *h, const char *filename, int flags);
    int (*url_read)(URLContext *h, unsigned char *buf, int size);
    int (*url_write)(URLContext *h, unsigned char *buf, int size);
    offset_t (*url_seek)(URLContext *h, offset_t pos, int whence);
    int (*url_close)(URLContext *h)
    struct URLProtocol *next; /* linked list로 관리한다 */
  }
- URLContext
  {
    struct   URLProtocol prot;
    int flags;
    int is_streamed; /*stream이 있다면 ture, default = false */
    int max_packet_size; /* stream의 길이 */
    void *priv_data; /*file descriptor를 저장한다*/
    char filename[1]; /*file이름*/
  }
- ByteIOContext
  {
    unsigned char *buffer; /* file에서 읽어온 data를 저장 */
    int buffer_size;
    unsigned char *buf_ptr, *buf_end;
    void *opaque; /*URLContext를 저장*/
    int (*read_packet)(void *opaque, uint8_t *buf, int buf_size);
    void (*write_packet)(void *opaque, uint8_t, int buf_size);
    int (*seek)(void *opaque, offset_t offset, int whence);
    offset_t pos; /* 현재 버퍼의 파일 포인터 위치 */
    int must_flush; /* eof_reached */
    int eof_reached; /* true if eof reached */
    int write_flag; /* file이 write를 위해 사용된다면, ture*/
    int is_streamed;
    int max_packet_size;
```

```
}
- AVFormatContext
{
    /* iformat과 oformat은 동시에 두값 모두 갖을 수 없다 */
    struct AVInputFormat *iformat;
    struct AVOutputFormat *oformat;
    void *priv_data;
    ByteIOContext pb;
    int nb_streams;
    AVStream *streams[MAX_STREAMS];
    char filename[1024]; /* input 또는 output 파일의 이름 */
    /* stream info */
    char title[512];
    char author[512];
    char copyright[512];
    char comment[512];
    char album[512];
    int year;  /* ID3 year, 0 if none */
    int track; /* track number, 0 if none */
    char genre[32]; /* ID3 genre */

    int flags; /* format의 종류를 저장 */
    /* pts를 제어하기 위한 변수 (직접 접근할 수 없다) */
    int pts_wrap_bits; /* pts에서 bit수 (특정 controller에 의해서 사용되어진다) */
    int pts_num, pts_den; /* 초를 계산하기 위한 변수 */
    /* 이 버퍼는 packets이 decode되지 않은 버퍼로 만들어졌을때 필요하다.
       mpeg의 codec parameters가 그러한 예이다. */
    struct AVPacketList *packet_buffer;

    /* decoding: position of the first frame of the component, in
       AV_TIME_BASE fractional seconds. NEVER set this value directly:
       it is deduced from the AVStream values. */
    int64_t start_time;
    /* decoding: duration of the stream, in AV_TIME_BASE fractional
       seconds. NEVER set this value directly: it is deduced from the
       AVStream values.  */
    int64_t duration;
    /* decoding: file의 size */
    int64_t file_size;
```

```
    /* decoding: 전체 stream bitrate(bit/s). file_size와 duration은
       ffmpeg이 자동으로 계산하기 때문에 직접 접근하지 않는다. */
    int bit_rate;
  }
- AVStream
  {
    int index;      /* stream index in AVFormatContext */
    int id;         /* format specific stream id */
    AVCodecContext codec; /* codec context */
    int r_frame_rate;       /* real frame rate of the stream */
    int r_frame_rate_base;/* real frame rate base of the stream */
    void *priv_data;
    /* internal data used in av_find_stream_info() */
    int codec_info_state;
    int codec_info_nb_repeat_frames;
    int codec_info_nb_real_frames;
    /* PTS generation when outputing stream */
    AVFrac pts;
    /* ffmpeg.c private use */
    int stream_copy; /* if TRUE, just copy stream */
    /* quality, as it has been removed from AVCodecContext and put in AVVideoFrame
     * MN:dunno if thats the right place, for it */
    float quality;
    /* decoding: position of the first frame of the component, in
       AV_TIME_BASE fractional seconds. */
    int64_t start_time;
    /* decoding: duration of the stream, in AV_TIME_BASE fractional
       seconds. */
    int64_t duration;
  }
- SDL_Overlay
  {
    uint32 format;
    int w, h;
    int planes;
    uint16 *pitches;
    uint8 **pixels;
    uint32 hw_overlay:1;
  }
```

- VideoPicture
  {
    double pts; /* presentation time stamp for this picture */
    SDL_Overlay *bmp;
    int width, height; /* source height & width */
    int allocated;
  }
- VideoState
  {
    SDL_Thread *parse_tid;
    SDL_Thread *video_tid;
    AVInputFormat *iformat;
    int no_background;
    int abort_request;
    int paused;
    int last_paused;
    AVFormatContext *ic;
    int dtg_active_format;

    int audio_stream;

    int av_sync_type;
    double external_clock; /* external clock base */
    int64_t external_clock_time;

    double audio_clock;
    double audio_diff_cum; /* used for AV difference average computation */
    double audio_diff_avg_coef;
    double audio_diff_threshold;
    int audio_diff_avg_count;
    AVStream *audio_st;
    PacketQueue audioq;
    int audio_hw_buf_size;
    /* samples output by the codec. we reserve more space for avsync compensation */
    uint8_t audio_buf[(AVCODEC_MAX_AUDIO_FRAME_SIZE * 3) / 2];
    int audio_buf_size; /* in bytes */
    int audio_buf_index; /* in bytes */
    AVPacket audio_pkt;
    uint8_t *audio_pkt_data;

```
    int audio_pkt_size;
    int64_t audio_pkt_ipts;

    int show_audio; /* if true, display audio samples */
    int16_t sample_array[SAMPLE_ARRAY_SIZE];
    int sample_array_index;
    int last_i_start;

    double frame_timer;
    double frame_last_pts;
    double frame_last_delay;
    double video_clock;
    int video_stream;
    AVStream *video_st;
    PacketQueue videoq;
    int64_t ipts;
    int picture_start; /* true if picture starts */
    double video_last_P_pts; /* pts of the last P picture (needed if B frames are present) */
    double video_current_pts; /* current displayed pts (different from
                                 video_clock if frame fifos are used) */
    int64_t video_current_pts_time; /* time at which we updated video_current_pts - used
                                       to have running video pts */
    VideoPicture pictq[VIDEO_PICTURE_QUEUE_SIZE];
    int pictq_size, pictq_rindex, pictq_windex;
    SDL_mutex *pictq_mutex;
    SDL_cond *pictq_cond;

    //      QETimer *video_timer;
    char filename[1024];
    int width, height, xleft, ytop;
  }
- AVInputFormat
  {
    const char *name;
    const char *long_name;
    /* size of private data so that it can be allocated in the wrapper */
    int priv_data_size;
    /* tell if a given file has a chance of being parsing by this format */
    int (*read_probe)(AVProbeData *);
```

```
    /* read the format header and initialize the AVFormatContext
       structure. Return 0 if OK. 'ap' if non NULL contains
       additionnal paramters. Only used in raw format right
       now. 'av_new_stream' should be called to create new streams.  */
    int (*read_header)(struct AVFormatContext *, AVFormatParameters *ap);
    /* read one packet and put it in 'pkt'. pts and flags are also
       set. 'av_new_stream' can be called only if the flag AVFMT_NOHEADER is used. */
    int (*read_packet)(struct AVFormatContext *, AVPacket *pkt);
    /* close the stream. The AVFormatContext and AVStreams are not
       freed by this function */
    int (*read_close)(struct AVFormatContext *);
    /* seek at or before a given pts (given in microsecond). The pts
       origin is defined by the stream */
    int (*read_seek)(struct AVFormatContext *, int64_t pts);
    /* can use flags: AVFMT_NOFILE, AVFMT_NEEDNUMBER, AVFMT_NOHEADER */
    int flags;
    /* if extensions are defined, then no probe is done. You should
       usually not use extension format guessing because it is not
       reliable enough */
    const char *extensions;
    /* general purpose read only value that the format can use */
    int value;
    /* private fields */
    struct AVInputFormat *next;
  }
```

1.4 함수별 분석

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

```
    {
      av_register_all(); /*AVInputFormat's mpegps_demux를 등록한다*/
      /*screen의 해상도를 측정한다*/
      stream_open() /*VideoState 구조체를 만든다*/
      event_loop() /*GUI에서 발생한 event를 수신한다*/
    }
```

1.4.2 int av_register_all()

```
    {
      /*MPEG파일 입력에 사용하는 AVInputFormat을 만든다*/
```

```
    mpegps_init();
}
```

1.4.3 static VideoState *stream_open(const char *filename,
AVInputFormat *iformat)

```
{
    VideoState *is;
    /*VideoState 구조체를 완성한다*/
    schedule_refresh(is, 40) /*화면 출력을 위한 timer를 등록한다*/
    is->parse_id = SDL_CreateThread(decode_thread, is);

    return is;
}
```

1.4.4 static int decode_thread(void *arg)

```
{
    av_open_input_file(AVFormatContext, ...);/*prepare to read a mpeg file*/
    stream_component_open(is, audio_index); /*video_thread를 시작한다*/
    stream_component_open(is, video_index); /*sdl_audio_callback을 시작한다*/
    av_read_packet(VideoState, AVPacket); /*AVPacket을 읽어온다*/
    packet_queue_put(VideoState->AVPacket, AVPacket); /*AVPacket을 저장한다.*/
    stream_component_close() /**/
    return 0;
}
```

1.4.5 int av_open_input_file(AVFormatContext **ic_ptr,
const char *filename,
int buf_size,
AVFormatParameters *ap)

```
{
    //Open a media file as input. The Codec ar not opend.
    //Only the file header (if present) is read.
    url_fopen(AVFormatContext->ByteIOContext, ...);

    return 0;
}
```

1.4.6 int url_fopen(ByteIOContext *s, const char *filename, int flags)

```
{
```

```
    URLContext *h;
    url_open(&h, ...);
    url_fopen(s, h);

    return 0;
}
```

1.4.7 int url_open(URLContext **puc, const char *filename, int flags)

```
{
    /*1. URLContext를 만듦*/
    puc->prot = URLProtocol; /*URLContext->URLProtocol 등록*/

    return 0;
}
```

1.4.8 int url_fdopen(ByteIOContext *s, URLContext *h)

```
{
    uint8_t buffer;
    /*buffersize 를 계산한다*/
    buffer = malloc(buffersize);

    init_put_byte(s, buffer, ..., h, url_read_packet, url_write_packet, url_seek_packet);

    return 0;
}
```

1.4.9 int init_put_byte(ByteIOContext *s, unsigned char *buffer,

```
        int buffer_size, int write_flag,
        void *opaque,
        int (*read_packet)(void *opaque, uint_8 *buf, int buf_size)
        int (*write_packet)(void *opaque, uint_8 *buf, int buf_size)
        int (*seek)(void *opaque, offset_t offset, int whence))
{
    /*ByteIOContext를 만든다.*/
    s->buffer = buffer;
    s->opaque = opaque; /*URLContext 저장*/
    s->read_packet = url_read_packet; /*각 함수 포인터를 저장한다*/
    s->write_packet = write_packet;
    s->seek = seek;
```

```
    return 0;
}
```

1.4.10 static int stream_component_open(VideoState *is, int stream_index)

```
{
    if(stream_index == AUDIO)
    {
        /* sdl_audio_callback 시작한다 */
    }
    else if(stream_index == VIDEO)
    {
        /* video_thread를 시작한다  */
    }
}
```

1.4.11 static void schedul_refresh(VideoState *is, int delay)

```
{
    /*add the refresh timer to draw the picture*/
     sdl_refresh_timer_cb -> SDL_PushEvent();
}
```

1.4.12 void event_loop(void)

```
 {
    //handle an evnet sent by the GUI
    SDL_Event event;
    for(;;)
    {
        SDL_WaitEvent(&evnet);
        switch(event.type)
        {
            case FF_REFRESH_EVENT:
                video_refresh_timer(event.user.data1);/*VideoState 전송*/
                break;
            default:
                break;
        }
    }
 }
```

1.4.13 int av_read_packet(AVFormatContext *s, AVPacket *pkt)

```
{
  AVPacketList *pktl;
  pktl = s->packet_buffer;
  /* av_register_all()에서 등록한
     read_packet(mpegps_read_packet()) 호출* /
  return s->iformat->read_packet(s, pkt);
}
```

1.4.14 static int mpegps_read_packet(AVFormatContext *s, AVPacket *pkt)

```
{
  get_byte(&s->pb);

  /*1. codec을 확인한다*/

  get_byte(&s->pb);

  /*2. 읽어온 data를 AVPacket으로 만든다*/

  /*3. AVStream->index 로 Audio 인지 Video인지 구별한다*/
  return 0;
}
```

1.4.15 int get_byte(ByteIOContext *s)

```
{
  if(buffer에 data가 있는가)
  {
    return *s->buf_ptr++;
  }
  else
  {
    fill_buffer(s);
  }
}
```

1.4.16 static void fill_buffer(ByteIOContext *s)

```
{
  /*init_put_byte()에서 등록한 read_packet()함수 호출*/
  len = s->read_packet(s->opaque, s->buffer, s->buffer_size);
```

```
    if( len<= 0)
    {
      s->eof_reached = 1;
    }
    else
    {
      s->pos + = len;
      s->buf_ptr = s->buffer;
      s->buf_end = s->buffer +  len;
    }
  }
```

1.4.17 static int packet_queue_put(PacketQueue *q, AVPacket *pkt)

```
  {
    /*VideoState->PacketQueue에 AVPacketList를 저장*/
    return 0;
  }
```

1.4.18 static void video_refresh_timer(void *opaque)

```
  {
    /*called to display each frame*/
    VideoState *is = (VideoState)opaque;

    if(VIDEO)
    {
      if(is->pictq_size == 0)
      {
        sehcule_refresh(is, 40)
      }
      else
      {
        /*VIDEO, AUDIO Sync를 계산*/
        schedule_refresh();
        video_display(is);
      }
    }
    else if(AUDIO)
    {
      sehcuel_refreah(is, 40);
```

```
        video_display(is); /*Audio 음파를 화면에 출력*/
      }
    }
```

1.4.19 static void video_display(VideoState *is)

```
    {
      //display picture
      if(AUDIO)
      {
        video_audio_display(is);
      }
      else if(VIDEO)
      {
        video_image_display(is);
      }
    }
```

1.4.20 static void video_image_display(VideoState *is)

```
    {
      VideoPicture *vp;
      vp = &is->pict[is->rindex];
      /*1. 코덱 해상도의를 구한다*/
      /*2. 화면 크기를 구한다*/

      /*VideoState->VideoPicture->SDL_Overlay 화면 출력*/
      SDL_DisplayYUVOverlay(vp, rect);
    }
```

1.4.21 static int video_thread(void *arg)

```
    {
      VideoState *is = arg;
      AVPacket pkt1, *pkt = &pkt1;
      unsigned char *ptr;
      uint64_t ipts;
      AVFrame frame;
      /*1. pause() 요구를 확인한다*/

      packet_queue_get(&is->videoq, pkt, 1);
      ipts = pkt->pts;
```

```
    ptr = pkt->data;

    /*2. 읽어온 AVPacket을 decode*/
    avcodec_decode_video(&is->video_st->codec,
                         &frame, &got_picture, ptr, len);
    output_picture2(is, &frame, pts);
  }
```

1.4.22
```
  static int packet_queue_get(PacketQueue *q, AVPacket *pkt, int block)
  {
    int ret;
    /* 1. packet_queue_put()에서 저장한 AVPacket을 읽는다.*/
    if(AVPacket 읽음)
      ret = 1;
    else
      ret = 0;

    return ret;
  }
```

1.4.23
```
  static int output_picture2(VideoState *is, AVFrame *src, double pts1)
  {
    /* 1. PTS를 계산 */

    queue_picture(is, src_frame, is->video_clock);
  }
```

1.4.24
```
  static int queue_picture(VideoState *is AVFrame *src,
                           doble block)
  {
    AVPicture pict;
    VideoPicture *vp;
    vp = &is->pictq[&is->windex];

    if(vp->bmp)
    {
      SDL_LockYUVOverlay();

      /*VideoState->VideoPicture->SDL_Overlay 영상정보입력*/
```

```
            pict.data[0] = vp->bmp->pixels[0];
            pict.data[1] = vp->bmp->pixels[1];
            pict.data[2] = vp->bmp->pixels[2];

            pict.linesize[0] = vp->bmp->pitches[0];
            pict.linesize[1] = vp->bmp->pitches[1];
            pict.linesize[2] = vp->bmp->pitches[2];

            img_convert(&pict, format, (AVPicture *)src_frame,
                        is->video_st->codec.pix_fmt,
                        is->video_st->codec.width,
                        is->video_st->codec.height);

            SDL_UnLockYUVOverlay();
        }
    }
```

1.5 참고문서

[1] ffmpeg, http://ffmpeg.sourceforge.net

2. video4linux 속도 향상을 위한 double buffering 방법

VideoForLinux에서 보다 빨리 읽어올 수 있도록 더블 버퍼링을 사용하는 방법에 대해서 기술한다. video4linux의 모든 부분을 기술하는 문서가 아니므로 video4linux의 초기 설정 부분은 video4linux문서를 참고 하기 바란다.

2.1 video4linux 초기화

video4linux 프로그래밍을 할 때 메모리에서 빨리 읽어오기 위해 메모리 맵을 사용해서 읽어오게 된다. 하지만, 메모리 맵만 가지고 1초에 30프레임을 캡쳐하는 것은 무리가 있다. 왜냐하면, video4linux에서는 이미지 한장의 캡쳐를 시작하면 캡쳐가 끝날때까지 sync를 하며 기다리게 되는데 여기서 sync로 인해 시간이 걸리게 된다. 따라서, 우리는 비디오 장치가 가지는 버퍼에서 더블 버퍼링을 사용해서 메모리에서 읽어오는 속도를 향상 시킬수 있다. 이런 속도 향상을 위해 더블버퍼를 사용하기 위해서 우리는 몇가지 초기화 작업을 해주어야 하는데 그 과정은 다음과 같다.

2.1.1 선언

video_mbuf와 video_mmap을 사용하기 위해서 두개의 구조체를 선언해 주어야 한다. 또한, 실제 영상을 넣어줄 *mframe과 버퍼의 frame숫자를 의미하는 fno를 선언한다.

```
static struct video_mbuf mbuf;
static struct video_mmap vmap;
static unsigned char *mframe;
static int fno;
```

2.1.2 버퍼의 정보 얻어 오기

버퍼를 사용하기 위해서는 ioctl을 사용해서 버퍼의 정보를 얻어온다.

```
  if(ioctl(fd,VIDIOCGMBUF,&mbuf)==-1){
    fprintf(stderr,"ioctl VIDIOCGMBUF: %s₩n",strerror(errno));
    exit(1);
  }
```

2.1.3 버퍼와 실제 저장할 char 포인터와 memory map

실제 데이터를 가져 가기 위하여 mmap함수를 사용해서 메모리를 맵핑해준다.

```
  if((mframe=(char*)mmap(0,mbuf.size,PROT_READ,MAP_SHARED,fd,0))==((void*)-1)){
    fprintf(stderr,"mmap: %s₩n",strerror(errno));
    exit(1);
  }
```

2.1.4 video_map 초기화

기본적인 width와 height, format을 설정하고 video_map의 vmap에 버퍼의 최대 프레임 수만큼을 실제 영상으로 채워 준다. 그리고, fno를 0으로 세팅한다.

```
  vmap.width=WIDTH;
  vmap.height=HEIGHT;
  vmap.format=VIDEO_PALETTE_YUV420P;
  for(fno=0;fno<mbuf.frames;fno++){
    vmap.frame=fno;
    if(ioctl(fd,VIDIOCMCAPTURE,&vmap)==-1){
      fprintf(stderr,"ioctl VIDIOCMCAPTURE: %s₩n",strerror(errno));
      exit(1);
  }
  fno = 0;
```

2.2 캡쳐

실제 영상으로 세팅되어있는 버퍼를 더블 버퍼링으로 사용하기 위해 다음과 같은 순서로 캡쳐하게 된다.
먼저 video4linux초기화에서 fno를 0으로 설정했기 때문에 vmap.frame에 0을 넣지 않고 fno를 넣는다. 그런후 VIDIOCSYNC를 통해 0번 프레임이 SYNC될 때 까지 기다린다. 벌써 채워져 있으므로 SYNC가 끝나게 되고 frame에 mframe의 위치에 mbuf.offsets[fno]만큼의 포인터를 더해 설정해 주게 된다. 그런후 fno를 vmap.frame에 다시 넣고 VIDIOCMCAPTURE를 통해 방금 SYNC한 프레임을 다시 캡쳐 하게 된다. 그리고 fno를 증가 시키고, 다시 이작업을 반복 하게 된다. 이때 fno가 최대 버퍼 싸이즈가 되면 fno=0으로 세팅하게 된다. 이렇게 하므로써 SYNC는 벌써 채워져 있는 버퍼를 SYNC하기 때문에 짧은 시간에 끝나거나 바로 끝나게 되고, 그동안 캡쳐 또한 SYNC하는 동안 블럭되어 있을 필요가 없기 때문에 캡쳐 속도가 빨라지게 된다.

```
---------------------------------------------------------------------
unsigned char *v4l_capture(int fd)
{
  unsigned char *frame;

  vmap.frame=fno;
  if(ioctl(fd,VIDIOCSYNC,&vmap)==-1){
    fprintf(stderr,"ioctl VIDIOCSYNC: %s₩n",strerror(errno));
    exit(1);
  }
#ifdef DEBUG
  //  fprintf(stderr,"ioctl VIDIOCSYNC:%d₩n",fno);
#endif

  frame=mframe+ mbuf.offsets[fno];
  vmap.frame=fno;
  if(ioctl(fd,VIDIOCMCAPTURE,&vmap)==-1){
    fprintf(stderr,"ioctl VIDIOCMCAPTURE: %s₩n",strerror(errno));
    exit(1);
  }
#ifdef DEBUG
  //  fprintf(stderr,"ioctl VIDIOCMCAPTURE:%d₩n",fno);
#endif

  fno++;
  if(fno>=mbuf.frames)
    fno=0;

  return frame;
}
---------------------------------------------------------------------
```

2.3 결론

결국 캡쳐 속도를 높일 수 있는 이유는 앞의 채워져 있는 버퍼를 SYNC를 해서 시간이 거의 안걸리고 SYNC된 버퍼를 먼저 캡쳐 하므로 현재 버퍼 캡쳐 시간에 캡쳐 하는 동안 SYNC는 다른 버퍼를 SYNC하게 된다. 따라서 현재 버퍼를 SYNC하기까지는 약간의 시간이 있기 때문에 SYNC때문에 버퍼 캡쳐를 블럭시키지 않아도 되므로 지연 시간이 거의 없게 된다.

3. bt848 및 bt878 칩셋이 장착된 TV 카드 시험

본 문서는 Redhat 9.0에서 bt848 및 bt878 칩셋이 장착된 TV 카드 7종을 시험한 결과서이다.

- Conexant bt848 및 bt878 칩셋이 장착된 TV 카드를 사용하기 위해서는 bttv 드라이버가 필요한데 이 드라이버는 Redhat 9.0의 경우 /lib/modules/2.4.20-8/kernel/drivers/media/video/bttv.o에 설치되어 있다.

- Redhat 9.0이 설치된 PC에 TV 카드를 장착한 후 리눅스를 부팅하면 TV 카드가 장착되었음이 인식되지만 bttv 드라이버를 포함한 필요한 드라이버가 자동으로 적재(insmod)되지는 않는다(주: 아래의 7개 TV 카드 실험 결과임). 이 때문에 다음과 같이 명령어 "modprobe"를 사용하여 필요한 드라이버를 적재하여야 한다.

- 명령어 "modprobe"를 사용할 때 어떤 TV 카드는 특별한 옵션 없이"modprobe bttv" 처럼 명령어을 실행해도 되지만 대부분의 TV 카드는 적절한 카드번호 및 튜너번호를 명시하는 옵션을 주어 "modprobe bttv card=카드번호 tuner=튜너번호" 처럼 명령어을 실행하여야 한다.

- 명령어 "modprobe bttv ..."를 실행하면 /lib/modules/2.4.20-8/modules.dep에정의된 드라이버 의존성 정의에 따라 다수의 드라이버가 같이 적재된다. 다음은 파일 modules.dep에서 bttv 드라이버에 대한 의존성 정의 부분만을 보인 것이다.

```
/lib/modules/2.4.20-8/kernel/drivers/media/video/bttv.o: ₩
        /lib/modules/2.4.20-8/kernel/drivers/i2c/i2c-core.o ₩
        /lib/modules/2.4.20-8/kernel/drivers/sound/soundcore.o ₩
        /lib/modules/2.4.20-8/kernel/drivers/i2c/i2c-algo-bit.o ₩
        /lib/modules/2.4.20-8/kernel/drivers/media/video/videodev.o
```

- 명령어 "modprobe bttv ..." 실행 후 "lsmod" 명령어를 실행해 보면 다음과 같은 드라이버가 적재되어 있음을 알 수 있다.

```
videodev          [bttv]
i2c-core          [tuner bttv i2c-algo-bit]
i2c-algo-bit      [bttv]
soundcore         [bttv]
bttv              (unused)
tuner             (autoclean)
```

- 위 드라이버가 적재되어 있으면 응용 프로그램 xawtv 혹은 tvtime 등을 실행하여 TV를 시청할 수 있다.

- 소리는 TV 카드의 스피커 단자에 연결된 스피커를 통하여 출력된다. 만약 이 단자가 사운드 카드의 입력으로 연결되어 있고 사운드 카드의 스피커 단자에 스피커가 연결되어 있다면 사운드 카드가 필요로 하는 드라이버가 설치되어 있어야 소리가 출력된다.

- 아래 자료는 국내 시판 중인 7개의 TV 카드에 대한 실험 결과로 얻은 카드번호 및 튜너번호이다.

모델이름: FusionPVR (http://www.dvico.co.kr/Products/FusionPVR.asp)
제조사명: 디비코(주) (http://www.dvico.co.kr)
카드번호: 0 (generic card)
튜너번호: 9

모델이름: PCTV
(http://www.pinnaclesys.com/Category.asp?Category_ID=5&Langue_ID=10)
제조사명: Pinnacle Systems, Inc. (http://www.pinnaclesys.com)
카드번호: 39
튜너번호: 33

모델이름: WinFast TV2000 XP (http://www.leadtek.com/multimedia.html)
제조사명: Leadtek Research, Inc. (http://www.leadtek.com)
카드번호: 34
튜너번호: 5
참고: 카드번호 및 튜너번호 주지 않고 modprobe 하여도 자동 처리됨

모델이름: SigmaTVII
(http://www.sigmacom.co.kr/system/html/tvc_SIGMA_TV_II.html)
제조사명: (주)시그마컴 (http://www.sigmacom.co.kr)
장치상태: 이상없음
카드번호: 44
튜너번호: 9

모델이름: Little OnAirTV (http://www.sasem.com/prod/main2-1.html)
제조사명: (주)사람과셈틀 (http://www.sasem.com)
카드번호: 43
튜너번호: 2, 8, 17, 21, 36

모델이름: Unitech TV (http://unitec.co.kr/product/tvcard/unitechtv.htm)
제조사명: (주)유니텍전자 (http://unitec.co.kr)
카드번호: 0 (generic card)

튜너번호: 2, 8, 17, 21, 36, 42

모델이름: AverTVStudio
(http://www.aver.com/products/tvtuner_AVerTV_studio.shtml)
제조사명: AVerMedia, Inc. (http://www.aver.com)
카드번호: 41
튜너번호: 2
참고: 카드번호 및 튜너번호 주지 않고 modprobe 하여도 자동 처리됨

- 참고자료

[1] bttv driver, http://linux.bytesex.org/v412/bttv.html.
[2] Eric Sandeen, The BTTV Mini-HOWTO v0.3 (http://metalab.unc.edu/LDP), February 2000.
[3] xawtv homepage, http://linux.bytesex.org/xawtv

4. 세가지 모션 검출 알고리즘

DVR Motion 부분에서 사용하는 모션 검출 알고리즘의 구현 방법에 대해서 기술한다.

4.1 개요

모션 검출 알고리즘 중에서 움직이는 객체를 찾아내는 방법으로는 프레임 차이법과 배경 차이법이 대표적인 방법이다. 프레임 차이법이란 연속된 프레임 내의 픽셀값의 차이가 임계값 이상이면 움직임이 발생했다고 판단하는 방법이다. 배경 차이법은 배경 프레임을 형성하고 현재 캡쳐된 프레임과 배경 프레임의 픽셀값의 차이로 인해 움직임을 검출하는 방식이다.

현재 DVR Motion 부분에서 사용할 알고리즘은 연속된 세 개의 프레임 차이로 움직임을 검출하는 3-프레임 차이법 알고리즘, 배경 모델과 현재 프레임의 픽셀값의 차이로 움직임을 검출하는 배경 차이법 알고리즘과 3-프레임 차이법과 적응 차이법을 합친 하이브리드(hybrid) 알고리즘 등 3가지 이다.

4.2 3-프레임 차이법

연속된 세 개의 프레임의 차이로 움직임을 검출한다.

4.2.1 알고리즘

3-프레임 차이법은 모션 검출 알고리즘 중에서 쉽게 구현할 수 있는 방법으로 연속적인 3개의 프레임을 이용하여 움직임을 검출하는 방법이다. 움직임이 발생했음은 아래의 식으로 판단한다.

(|I_n(X) - I_n-1(X)| > Threshold) && (|I_n(X) - I_n-2(X)| > Threshold)

즉, 현재 프레임의 픽셀값(I_n(X))과 한 단계 이전 프레임의 픽셀값(I_n-1(X))의 차가 미리 지정된 임계값(Threshold)보다 크고 현재 프레임의 픽셀값(I_n(X))과 두 단계 이전 프레임의 픽셀값(I_n-2(X))의 차가 같은 임계값보다 크면 움직임이 발생했다고 판단한다. 간편하게 구현할 수 있는 것에 반해 좋은 효과를 가지는 알고리즘이다.

4.2.2 구현 방법

```
-----------------------------------------------------------------------
three_frame_diff(unsigned char *frame)
{
            ...
  static int n=2,n_1=1,n_2=0;
```

```
  static unsigned char frame_tmp[3][WIDTH*HEIGHT];
  int motion=0;
          ...
  for(y=0;y<HEIGHT;y++){
    for(x=0;x<WIDTH;x++){
      i=(y*WIDTH+x);
      if(((abs(frame_tmp[n][i]-frame_tmp[n_1][i])>MOTION_THRESHOLD)&&
          (abs(frame_tmp[n][i]-frame_tmp[n_2][i])>MOTION_THRESHOLD))){
            ...
        motion++;
      }
    }
  }
  if(motion>MOTION_AREA){
          ...
  }
}
```

현재 프레임, 한 단계 이전 프레임, 두 단계 이전 프레임을 저장하기 위해서 frame_tmp[3][WIDTH*HEIGHT]을 선언한다(하지만 모션 검출에 쓰이는 성분은 yuv 포맷의 y 성분만을 이용한다. 따라서 WIDTH*HEIGHT만 필요하다). 모션 검출 알고리즘을 적용한다. (WIDTH과 HEIGHT는 캡쳐 영상의 가로길이와 세로길이를 뜻한다). 세 프레임의 픽셀값의 차로 움직임이 발생했음을 판단한다. 움직임이 발생하면 움직임이 발생한 픽셀의 개수를 저장하는 motion 변수의 값을 증가시킨다. 최종적으로 한 프레임 내에서 모션이 발생한 픽셀의 수가 미리 정의된 픽셀의 수(MOTION_AREA) 보다 많아야 모션이 발생했다고 판단한다.

4.3 배경 차이법

배경 프레임을 구성한 뒤 현재 캡쳐된 프레임과 배경 프레임의 픽셀값의 차로 움직임을 검출한다.

4.3.1 알고리즘

일정 시간 동안 각 픽셀당 배경 프레임을 구성한 뒤, 배경 프레임의 픽셀값과 현재 프레임의 픽셀값의 차가 특정 임계값보다 크면 움직임이 발생했다고 판단한다. 하지만 배경 차이법 알고리즘은 처음에 각 픽셀의 배경 프레임을 구성할 때, 움직이는 객체가 없어야 한다는 단점을 가지고 있다. 아래의 식으로 인해 움직임이 발생했음을 판단한다.

|B_n(X) - I_n(X)| > Threshold

현재 캡쳐되는 영역에서 많은 객체들의 움직임이 발생하고, 카메라 영역 내에서 새로운 객체가 나타나 오랜 기간 멈춘 상태로 있다면 배경으로 인식하기 위해서 배경 프레임을 업데이트 시켜야 하는데 배경 프레임을 업데이트 시키는 식이 아래에 제시되어 있다(사람이 새로운 컴퓨터를 카메라 영상 내에 두고 다른 곳으로 이동했을 때, 일정 시간이 지난 후에는 컴퓨터를 배경으로 인식해야한다). 이 알고리즘은 오랜 기간 정지하고 있던 단색의 객체가 다시 움직이기 시작할 때, 객체의 경계 부분만 인식되는 “구멍(hole)”현상 발생하는 단점이 있다.

B_n(X) = aB_n-1(X) + (1-a)I_n(X) (움직임이 발생했을 때)
= B_n-1(X) (움직임이 발생하지 않았을 때)

4.3.2 구현 방법

```
---------------------------------------------------------------------
background_subtraction(unsigned char *frame)
{
          ...
  static unsigned char frame_tmp[WIDTH*HEIGHT];
  static unsigned char frame_bg[WIDTH*HEIGHT];
  int motion=0,moved=0;
          ...
  for(y=0;y<HEIGHT;y++){
    for(x=0;x<WIDTH;x++){
      i=(y*WIDTH+ x);
      if(abs(frame_bg[i]-frame_tmp[i])>MOTION_THRESHOLD){
        moved=1;
         ...
        motion++;
      }
      if(moved==1)
      {
        frame_bg[i]=frame_tmp[i];
      } else {
        frame_bg[i]=ALPHA*frame_bg[i]+(1-ALPHA)*frame_tmp[i];
      }
    }
  }
  if(motion>MOTION_AREA){
```

```
  ...
  }
}
```

배경 프레임을 위해서 frame_bg[WIDTH*HEIGHT]를 선언한다. moved 변수는움직임의 발생 유무를 판단해주는 플래그로 사용하는 변수이다. 배경 프레임과 현재 프레임의 픽셀값의 차로 움직임을 판단한다. 움직임이 발생하면 motion 변수의 값을 증가시킨다. 최종적으로 한 프레임 내에서 모션이 발생한 픽셀의 수가 미리 정의된 픽셀의 수(MOTION_AREA) 보다 많아야 움직임이 발생했다고 판단한다.

4.4 하이브리드(hybrid)

3-프레임 차이법과 적응 차이법을 조합해서 움직임을 검출한다.

4.4.1 알고리즘

적응 배경 차이법과 3-프레임 차이법을 결합한 방법으로 3-프레임 차이법을 이용하여 움직임의 발생 여부를 판단한다. 판단 방법은 현재 프레임(n)과 직전 프레임(n-1)과의 픽셀값의 차가 임계값보다 크고 현재 프레임(n)과 그 직전 프레임(n-2)과의 픽셀값의 차가 임계값보다 크면 움직임이 발생했다고 판단한다. 움직임이 발생했음은 아래의 식으로 판단한다.

(|I_n(X) - I_n-1(X)| > Threshold) && (|I_n(X) - I_n-2(X)| > Threshold)

하지만 위 식의 문제점은 같은 픽셀 정보를 가지는 객체가 움직였을 때, 객체의 경계에 있는 픽셀는 움직임이 발생했다고 식별이 되지만 객체 내부에 있는 픽셀들은 값이 변했지만 값이 변하지 않았다고 인식된다는 점이다. 즉, 특정 위치의 픽셀이 객체 내에서 가리키는 위치는 변했지만 픽셀값이 일치하기 때문에 움직임으로 인식되지 못하는 문제점이 발생한다. 따라서 이를 보완해야 한다. 움직인 객체의 경계 안에 있는 모든 픽셀값에 대해 아래의 식을 적용하면 움직이는 객체의 내부에 대한 처리를 할 수 있다.

b_n= {x : |I_n(x) - B_n(x)| > Tn(x), x ∈ R}

또한 이 알고리즘에서는 배경 프레임 B_n(x)과 임계값 T_n(x)을 새로운 프레임이 들어올 때마다 업데이트하도록 한다.

B_n(X) = aB_n-1(X) + (1-a)I_n(x) (움직임이 발생하지 않을 때)
= B_n-1(X) (움직임이 발생할 때)

T_n(X) = aT_n(X) + (1-a)(5*| I_n(X) - B_n(X)|)(움직임이 발생하지 않을 때)
= T_n(X) (움직임이 발생할 때)

4.4.2 구현 방법

```
---------------------------------------------------------------------
hybrid(unsigned char *frame)
{
        ...
  static unsigned char frame_tmp[3][WIDTH*HEIGHT];
  static unsigned char frame_bg[3][WIDTH*HEIGHT];
  static unsigned char ts[3][WIDTH*HEIGHT];
  int motion=0,moved=0;
        ...
  for(y=0; y<HEIGHT; y++){
    for(x=0; x<WIDTH; x++){
      i=(y*WIDTH+ x);
      if (((abs(frame_tmp[n][i]-frame_tmp[n_1][i])>ts[n][i]) &&
          (abs(frame_tmp[n][i]-frame_tmp[n_2][i])>ts[n][i]))){
        moved=1;
        ...
        motion++;
      }
      if(abs(frame_tmp[n][i]-frame_bg[n][i])>ts[n][i])
      {
        if(moved==0)
        {
          moved=1;
          motion++;
        }
      }
      if(moved==1){
        frame_bg[n_2][i]=frame_bg[n][i];
        ts[n_2][i]=ts[n][i];
      } else {
        frame_bg[n_2][i]=ALPHA*frame_bg[n][i]+ ALPHA*frame_tmp[n][i];
        ts[n_2][i]=ALPHA*ts[n][i]+ (1-ALPHA)*(5*abs(frame_tmp[n][i]-
frame_bg[n][i]));
      }
```

```
    }
  }
  if(motion>MOTION_AREA){
  ...
  }
}
```

배경 프레임을 위해서 frame_bg[3][WIDTH*HEIGHT]를 선언한다. moved 변수는 움직임의 발생 유무를 판단해주는 플래그로 사용하는 변수이다. 최우선 3-프레임 차이법으로 움직임의 발생 유무를 판단한다. 움직임이 발생했을 때는 motion 변수의 값을 증가시킨다. 일차적으로 움직임을 검출한 상태에서 배경 프레임과 현재 입력 프레임의 픽셀값의 차를 이용하여 다시 한번 움직임을 검출한다. 이때 3-프레임 차이법에서는 움직임으로 판단하지 못했던 픽셀이 있다면 모션이 발생했다고 표시하고 motion 변수의 값을 다시 증가시킨다. 그런 후에 상황에 맞춰서 배경 프레임과 임계값을 업데이트한다. 최종적으로 한 프레임 내에서 모션이 발생한 픽셀의 수가 미리 정의된 픽셀의 수(MOTION_AREA) 보다 많아야 움직임이 발생했다고 판단한다.

4.5 참고문서

[1] Robert T. Collins, Alan J. Lipton, Takeo Kanade, Hironobu Fujiyoshi, David Duggins, Yanghai Tsin, David Tolliver, Nobuyoshi Enomoto, Osamu Hasegawa, Peter Burt and Lambert Wixson, A System for Video Surveillance and Monitoring: VSAM Final Report, Technical report CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, May, 2000.

[2] 신동하, 이우철, Video4Linux를 이용한 모션 검출 알고리즘의 구현, "소프트웨어 미디어 연구소", 상명대학교, 2004.

제6장 설치 및 사용 방법

본 장에서는 본 연구 결과인 "리눅스 디지틀 비디오 레코더" 소프트웨어에 대해 기술한다. 1절에서는 DVR TV의 설치 및 사용 방법을 설명하고 수행화면을 보여준다. 2절에서는 DVR Motion의 설치 및 사용 방법을 설명하고 수행화면을 보여준다.

제1절 DVR TV 설치 및 사용 방법

본 절에서는 DVR TV 설치 및 사용 방법에 대해 기술하고, 수행 화면을 보여준다.

1. 설치방법

DVR TV 패키지는 현재 필요한 다른 소프트웨어 패키지와 시스템이 준비되어 있다는 가정하에 정상적으로 작동하도록 작성되었다.
DVR TV 소스 패키지를 컴파일하기 위해서는 ffmpeg 소프트웨어 패키지가 컴파일하는 시스템에 설치되어 있어야 한다. ffmpeg이 설치되어있다는 가정하에 DVR TV 컴파일을 설명한다. 구체적인 명령은 다음과 같다.

예) $ tar xvzf dvrtv-0.x.x.tar.gz
$ cd dvrtv-0.x.x
$ make

컴파일이 다 되었다면 다음과 같은 실행파일이 생성된다.

- dvrtv-save: 일정분량의 방송을 버퍼에 저장하는 프로그램
- dvrtv-play: 버퍼에 저장된 방송을 화면으로 출력하는 프로그램
- dvrtv-reservation: 녹화할 방송을 예약관리하는 프로그램
- dvrtv-recorder: 예약한 방송을 동영상 파일로 압축저자하는 프로그램

각자의 취향에 따라 적당한 디렉토리에 실행파일들을 복사하고 사용하면 된다.
정상적인 실행을 위해서는 TV 카드 드라이버 모듈과 사운드 카드 드라이버 모듈이 커널에 등록되어 있어야 한다. 본 프로그램이 사용하는 v4l(TV 카드) 디바이스 파일은 /dev/video0이고, OSS(사운드 카드) 디바이스 파일은 /dev/dsp 이다. 만약 위 디바이스 파일이 없다면 모든 모듈이 커널에 정상적으로 등록되어 있다고 하더라도 프로그램이 정상적으로 실행되지 않을 것이다. 그럴경우 mknod 명령어를 이용하여 디바이스 파일을 생성하라.
예) $ mknod /dev/dsp c 14 3

2. 사용법

2.1 조작 키

dvrtv-play에서 사용되는 조작키는 다음과 같다. 다음에서 알파벳 키는 대문자 소문자를 구분하지 않는다.

'Q': 종료
'E': 대기 모드
'T': TV 출력
'D': 녹화된 파일 열기 (decode)
'P': 일시정지 (pause)
'S': 슬로우모션 (slow motion)
방향키(상): 볼륨 조절 (소리 키움)
방향키(하): 볼륨 조절 (소리 줄임)
방향키(좌): 채널 변경 (채널 내림)
방향키(우): 채널 변경 (채널 올림)
Shift+ 방향키(좌): 타임 쉬프팅 (앞으로 이동)
Shift+ 방향키(우): 타임 쉬프팅 (뒤로 이동)

2.2 조작 예

2.2.1 dvrtv-save 실행

dvrtv-save 프로세스 실행은 다음과 같이 한다.

예) $ dvrtv-save /dev/video0 /dev/dsp save.buf
(1) (2) (3) (4)
(1)dvrtv-save 실행 파일
(2)TV 수신 장치 파일
(3)사운드 출력 장치 파일
(4)TV 버퍼 파일

2.2.2 dvrtv-play 실행

dvrtv-play 프로세스 실행은 다음과 같이 한다..

예) $ dvrtv-play save.buf mpeg2raw.buf
(1) (2) (3)

(1)dvrtv-play 실행 파일
(2)TV 버퍼 파일
(3)decode 버퍼 파일

2.2.3 TV 시청과 종료

TV를 시청하고 종료하기 위해서는 다음과 같은 과정을 거친다.

- dvrtv-save가 실행되고 있는 상태로 dvrtv-play를 실행한다.
- dvrtv-play가 실행되면 대기상태이다.
- 'T'키를 눌러 TV를 출력한다.
- 'Q'키를 눌러 종료한다.

2.2.4 녹화된 파일 재생과 종료

녹화된 파일을 재생하고 종료하기 위해서는 다음과 같은 과정을 거친다.

- dvrtv-play를 실행한다.
- dvrtv-play가 실행되면 대기상태이다.
- 'D'키를 눌러 decode한 mpeg파일(녹화된 파일)을 출력한다.
- 'Q'키를 눌러 종료한다.

2.2.5 TV 시청과 녹화된 파일 재생 바꾸기

TV 시청 중 녹화된 파일을 재생하고 싶을 때 또는 녹화된 파일을 재생 중 TV를 시청하고 싶을 때 다음과 같은 과정을 거친다.

- dvrtv-save가 실행되고 있는 상태로 dvrtv-play를 실행시킨다.
- 'T'키를 눌러 TV를 실행시킨다.('D'키를 눌러 decode를 실행시킨다.)
- 'E'키를 눌러 대기 상태로 나간다.
- 'D'키를 눌러 decode를 실행시킨다.('T'키를 눌러 TV를 실행시킨다.)

2.2.6 TV 시청 중 볼륨 조절, 채널 변경, 타임 쉬프팅, 슬로우모션, 일시정지

TV 시청 중 볼륨 조절, 채널 변경, 타임 쉬프팅, 슬로우모션, 일시정지를 하기 위해서는 다음과 같은 과정을 거친다.
- dvrtv-save가 실행되고 있는 상태로 dvrtv-play를 실행한다.
- dvrtv-play가 실행되면 대기상태이다.
- 'T'키를 눌러 TV를 출력한다.

2.2.6.1 볼륨 조절

다음과 같이 볼륨 조절을 한다.

- 방향키(상)을 눌러 볼륨을 키운다.
- 방향키(하)를 눌러 볼륨을 줄인다.

2.2.6.2 채널 변경

다음과 같이 채널 변경을 한다.

- 방향키(좌)를 눌러 채널을 내린다.
- 방향키(우)를 눌러 채널을 올린다.

2.2.6.3 타임 쉬프팅

다음과 같이 타임 쉬프팅을 한다.

- Shift+ 방향키(좌)를 눌러 앞으로 이동해 지나간 부분을 다시 본다.
- Shift+ 방향키(우)를 눌러 뒤로 이동해 원래 위치로 돌아간다.

2.2.6.4 슬로우모션

TV 시청 중 다음과 같이 슬로우모션을 하면 10초 전으로 돌아가서 느린화면으로 10초동안 보여준다.

- 'S'를 눌러 슬로우모션을 한다.

2.2.6.5 일시정지

다음과 같이 일시정지를 한다.

- 'P'를 눌러 일시정지를 한다.

2.2.7 녹화된 파일 재생 중 볼륨 조절, 타임 쉬프팅, 슬로우모션, 일시정지

녹화된 파일 재생 중 볼륨 조절, 타임 쉬프팅, 슬로우모션, 일지정지를 하기 위해서는 아래와 같은 과정을 거친다.

- dvrtv-play를 실행한다.
- dvrtv-play가 실행되면 대기상태이다.
- 'D'키를 눌러 decode한 mpeg파일을 출력한다.

2.2.7.1 볼륨 조절

다음과 같이 볼륨 조절을 한다.

- 방향키(상)을 눌러 볼륨을 키운다.
- 방향키(하)를 눌러 볼륨을 줄인다.

2.2.7.2 타임 쉬프팅

다음과 같이 타임 쉬프팅을 한다.

- Shift+ 방향키(좌)를 눌러 앞으로 이동해 지나간 부분을 다시 본다.
- Shift+ 방향키(우)를 눌러 뒤로 이동해 원래 위치로 돌아간다.

2.2.7.3 슬로우모션

재생 중 다음과 같이 슬로우모션을 하면 10초 전으로 가서 느린화면으로 10초동안 보여준다.

- 'S'를 눌러 슬로우모션을 한다.

2.2.7.4 일시정지

다음과 같이 일시정지를 한다.

- 'P'를 눌러 일시정지를 한다.

2.2.8 dvrtv-recorder 실행

dvrtv-recorder 프로세스 실행은 다음과 같이 한다.

예) $ dvrtv-recorder record.mpeg save.buf
(1) (2) (3)
(1)dvrtv-recorder 실행 파일
(2)mpeg 파일
(3)TV 버퍼 파일

2.2.9 dvrtv-reservation 실행

dvrtv-reservation 프로세스 실행은 다음과 같이 한다.

예) $ dvrtv-reservation 20040726/2100-20040726/2150-09-news
(1) (2)
(1)dvrtv-reservation 실행 파일
(2)시작년월일/시분-종료년월일/시분-채널-프로그램정보

3. 수행 화면

수행 화면에서는 DVR TV 소프트웨어의 수행되는 화면들을 제시하고 수행화면에 대해 간략히 설명한다. 수행 화면 부분에서 제시하는 화면은 크게 TV 시청시 기능별 수행 화면과 녹화된 파일 재생시 기능별 수행 화면으로 나눈다.

3.1 TV 시청시 기능별 수행 화면

TV 시청시 기능별 수행 화면에는 TV 시청 화면, 볼륨 조절 화면, 타임 쉬프팅 화면, 슬로우모션 화면, 일시정지 화면이 있다.

3.1.1 TV 시청 화면

<그림 6.1> TV 시청 화면

그림 6.1은 TV 시청 중인 화면이다.

3.1.2 TV 시청시 볼륨 조절 화면

다음은 TV 시청시 볼륨 조절 기능 관련 화면이다.

<그림 6.2> TV 시청시 볼륨 조절 전 화면

<그림 6.3> TV 시청시 볼륨 조절 후 화면

그림 6.2는 볼륨을 조절하기 전 화면으로 볼륨은 60으로 되어있다. 볼륨 조절은 방향키를 한 번 누를 때 +5(or -5)씩 된다. 그림 6.3은 방향키(위)를 3번 눌러 볼륨을 75로 조절한 것이다.

3.1.3 TV 시청시 채널 변경 화면

다음은 TV 시청시 채널 변경 기능 관련 화면이다.

<그림 6.4> TV 시청시 채널 변경 전 화면

<그림 6.5> TV 시청시 채널 변경 후 화면

그림 6.4는 채널을 조절하기 전 화면으로 채널은 11번이다. 채널 변경은 방향키를 한 번 누를 때 +1(or -1)씩 된다. 그림 6.5는 방향키(좌)를 4번 눌러 채널을 7번으로 변경한 것이다.

3.1.4 TV 시청시 타임 쉬프팅 화면

다음은 TV 시청시 타임 쉬프팅 기능 관련 화면이다.

<그림 6.6> TV 시청시 타임 쉬프팅 전 화면

<그림 6.7> TV 시청시 타임 쉬프팅 후 화면

그림 6.6은 타임 쉬프팅하기 전 화면으로 우축하단에 실시간을 나타내는 'LIVE' 표시가 나타난다. 그림 6.7은 타임 쉬프팅한 후 화면으로 방향키(좌)를 2번 눌러 20초 전의 내용을 보고 있는 것이다.

3.1.5 TV 시청시 슬로우모션 화면

다음은 TV 시청시 슬로우모션 기능 관련 화면이다.

<그림 6.8> TV 시청시 슬로우모션 전 화면

<그림 6.9> TV 시청시 슬로우모션 후 화면

그림 6.8은 슬로우모션 시작하기 전 화면이다. 그림 6.9는 'S'키를 눌러서 10초전으로 이동하여 천천히 화면을 보여주는 것이다. 우축 상단에 'slow motion' 표시가 나타난다.

3.1.6 TV 시청시 일시정지 화면

다음은 TV 시청시 일시정지 기능 관련 화면이다.

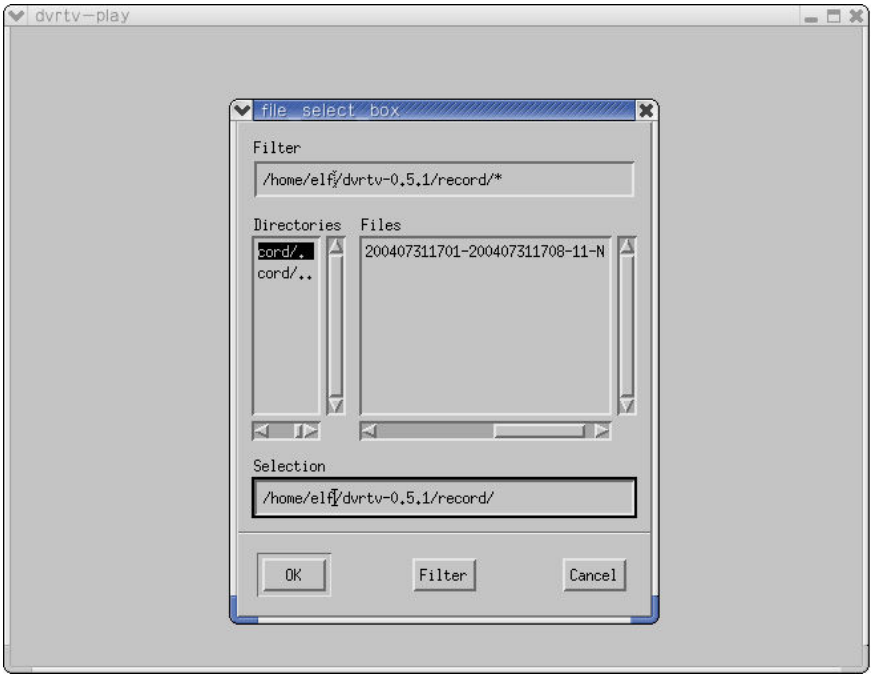
<그림 6.10> TV 시청시 일시정지 화면

그림 6.10은 'P'키를 눌러 일시정지한 화면을 보여준다. 우측 상단에 'pause'라는 표시가 나타난다.

3.2 녹화된 파일 재생시 기능별 수행화면

녹화된 파일 재생시 기능별 수행 화면에는 녹화된 파일 재생 화면, 볼륨 조절 화면, 타임 쉬프팅 화면, 슬로우모션 화면, 일시정지 화면이 있다.

3.2.1 녹화된 파일 재생 화면



<그림 6.11> 녹화된 파일 열기 화면

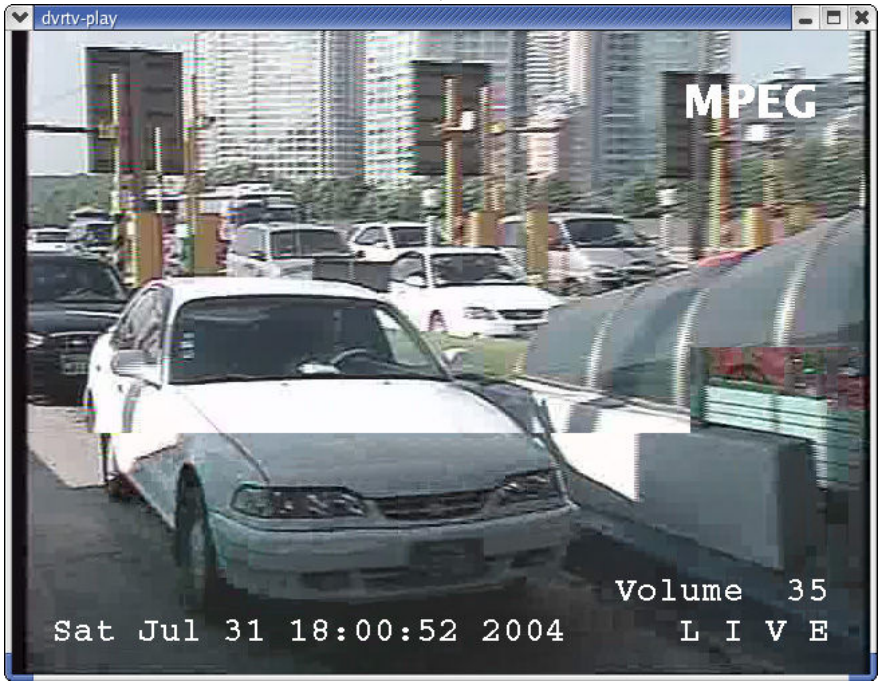
<그림 6.12> 녹화된 파일 재생 화면

그림 6.11은 'D'키를 눌러 녹화된 파일을 재생하기 위해 파일을 선택하는 화면이다. 그림 6.12는 녹화된 파일 재생 화면으로, 우측 상단에 'MPEG' 표시가 나타난다.

3.2.2 녹화된 파일 재생시 볼륨 조절 화면

다음은 녹화된 파일 재생시 볼륨 조절 기능 관련 화면이다.

<그림 6.13> 녹화된 파일 재생시 볼륨 조절 전 화면



<그림 6.14> 녹화된 파일 재생시 볼륨 조절 후 화면

그림 6.13은 볼륨을 조절하기 전 화면으로 볼륨은 75로 되어있다. 그림 6.14는 방향키(아래)를 8번 눌러 볼륨을 35로 조절한 것이다.

3.2.3 녹화된 파일 재생시 타임 쉬프팅 화면

다음은 녹화된 파일 재생시 타임 쉬프팅 기능 관련 화면이다.

<그림 6.15> 녹화된 파일 재생시 타임 쉬프팅 전 화면

<그림 6.16> 녹화된 파일 재생시 타임 쉬프팅 후 화면

그림 6.15는 타임 쉬프팅하기 전 화면으로 우축하단에 실시간을 나타내는 'LIVE' 표시가 나타난다. 그림 6.16은 타임 쉬프팅한 후 화면으로 방향키(좌)를 4번 눌러 40초 전의 내용을 보고 있는 것이다.

3.2.4 녹화된 파일 재생시 슬로우모션 화면

다음은 녹화된 파일 재생시 슬로우모션 기능 관련 화면이다.

<그림 6.17> 녹화된 파일 재생시 슬로우모션 전 화면

<그림 6.18> 녹화된 파일 재생시 슬로우모션 후 화면

그림 6.17은 슬로우모션 시작하기 전 화면이다. 그림 6.18은 'S'키를 눌러서 10초전으로 이동하여 천천히 화면을 보여주는 것이다. 우축 상단에 'slow motion' 표시가 나타난다.

3.2.5 녹화된 파일 재생시 일시정지 화면

다음은 녹화된 파일 재생시 일시정지 기능 관련 화면이다.

<그림 6.19> 녹화된 파일 재생시 일시정지 화면

그림 6.19는 'P'키를 눌러 일시정지한 화면을 보여준다. 우측 상단에 'pause'라는 표시가 나타난다

제2절 DVR Motion 설치 및 사용 방법

본 절에서는 DVR Motion의 설치 및 사용 방법을 설명하고 수행화면을 보여준다.

1. 설치 방법

DVR Motion(dvrmotion) 프로그램 압축을 푼 후 dvrmotion.conf를 자신이 원하는 대로 수정한다. 그 후 쉘에서 make를 수행하고, 수퍼 유저 권한을 갖은 상태에서 make install을 한다. make install 후 /usr/local/dvrmotion 폴더에 설치된다. 이 때, pwcx 모듈이 올라와 있어야하고, ffmpeg 프로그램이 설치되어있어야하고, apache가 실행되고 있어야 한다.
다음은 예를 들어 설명한다.

- $ tar zxvf dvrmotion-0.4.tar.gz
- $ cd dvrmotion-0.4
- $ emacs dvrmotion.conf /* dvrmotion.conf 수정-IP,TITLE */
- $ make
- $ make install /* 수퍼 유저 권한 있어야 함 */

2. 사용 방법

dvrmotion 프로그램을 설치 후 다음과 같이 수행시킨다.
- $ /usr/local/dvrmotion/dvrmotion start /* 수퍼 유저 권한 있어야 함 */

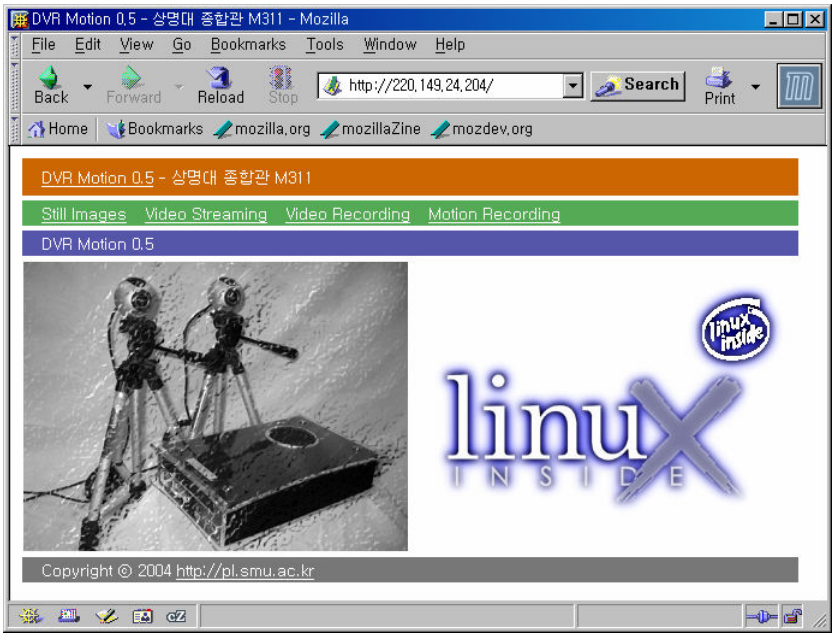
dvrmotion 프로그램의 수행을 멈추려면 다음과 같이 한다.
- $ /usr/local/dvrmotion/dvrmotion stop /* 수퍼 유저 권한 있어야 함 */

dvrmotion 프로그램이 수행되면, 웹에서 현재 기록중인 동영상을 볼 수 있을 뿐만 아니라, 저장 된 동영상과 모션만 저장된 동영상도 볼 수 있다. 사용법은 수행화면을 참조하기 바란다.

3. 수행 화면

수행 화면에서는 DVR Motion 소프트웨어의 수행되는 화면들을 제시하고 수행화면에 대해 간략히 설명한다. 수행화면 부분에서 제시하는 화면은 DVR Motion 웹 페이지의 메인(main) 화면, 정지 화면(still images 부분), 비디오 스트리밍 화면(video streaming 부분), 전체 저장된 영상 날짜별 목록(video recording 부분), 전체 저장된 영상 시간대별 목록, 그리고 모션(motion)만 저장된 영상 시간대별 목록이다. 모션만 저장된 영상 날짜별 목록(motion recording 부분)은 전체 저장된 영상 날짜별 목록과 비슷하므로 관련 수행 화면은 제시하지 않는다.

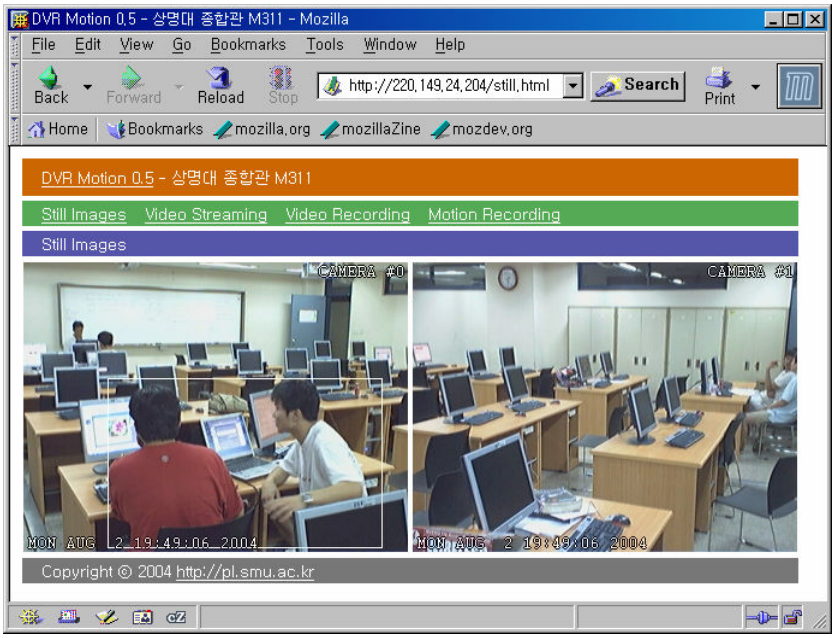
3.1 DVR Motion 웹 페이지 메인(main) 화면



<그림 6.20> DVR Motion 웹 페이지 메인 화면

그림 6.20은 DVR Motion 웹 페이지 메인 화면이다. DVR Motion은 웹 서비스을 함으로써 사용자들에게 보다 편한 인터페이스를 제공한다.

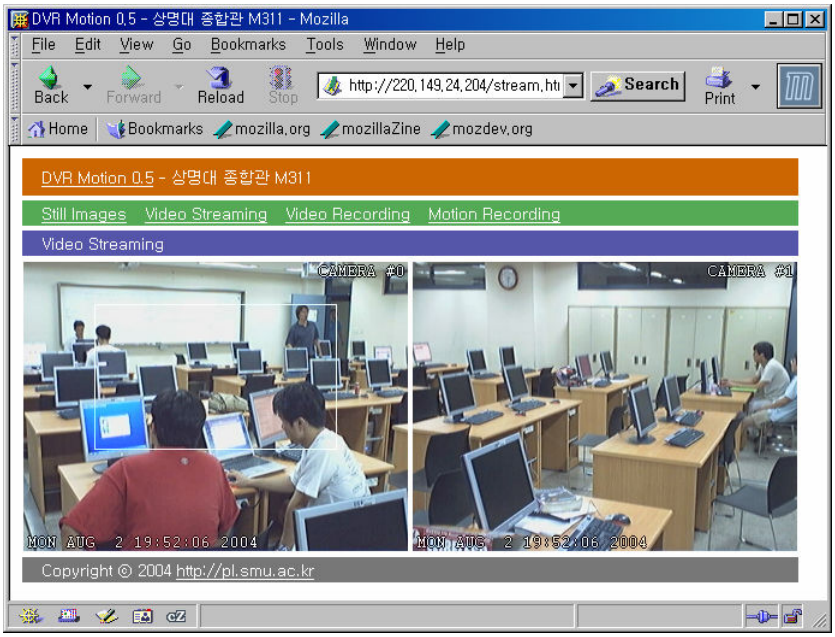
3.2 정지 화면



<그림 6.21> 정지 화면

그림 6.21은 DVR Motion에서 정지화면을 보여준다. DVR Motion 웹 페이지의 "Still Images" 메뉴를 클릭했을 때 보여지며, 현재 한 화면만을 보여준다.

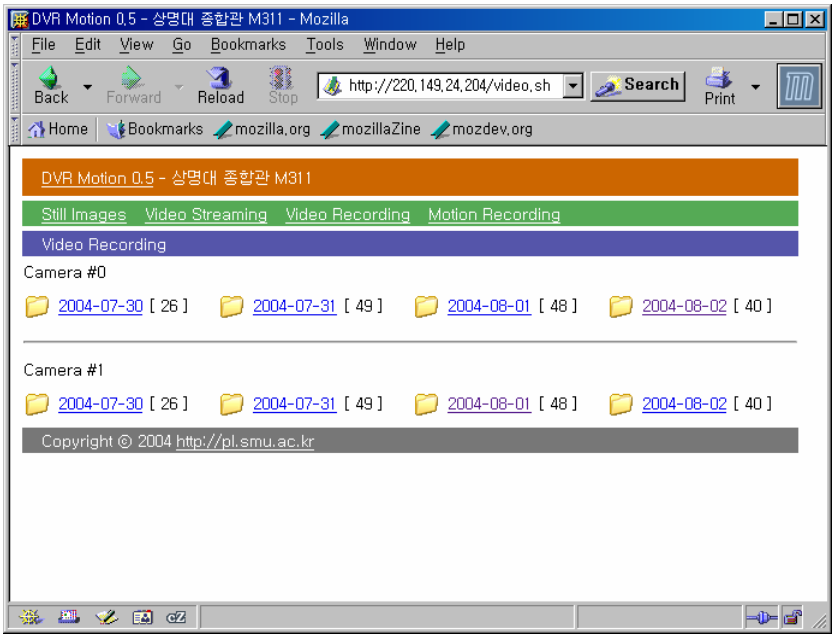
3.3 비디오 스트리밍(streaming) 화면



<그림 6.22> 비디오 스트리밍 화면

그림 6.22는 DVR Motion의 비디오 스트리밍 화면을 보여준다. DVR Motion 웹 페이지의 "Video Streaming" 메뉴를 클릭했을 때 보여지며, 클라이언트에게 움직임(motion)이 검출된 동영상이 스트리밍되는 것을 볼 수 있다. 흰 박스 부분은 움직임이 검출된 영역이다.

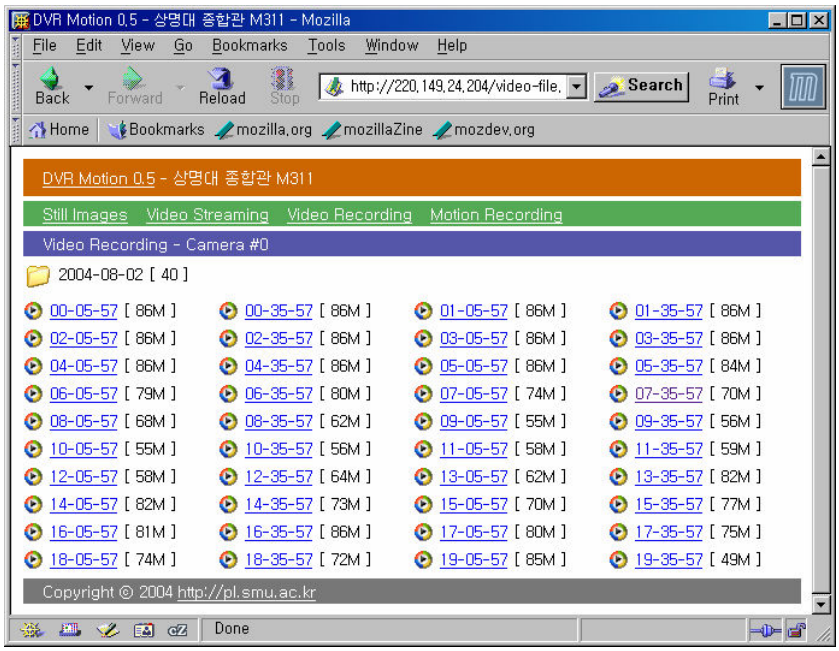
3.4 전체 저장된 영상 날짜별 목록



<그림 6.23> 전체 저장된 영상 날짜별 목록

그림 6.23은 전체 저장된 영상을 날짜별로 보여준다. DVR Motion 웹 페이지의 "Video Recording" 메뉴를 클릭했을 때 보여지며, DVR Motion 에서는 전체 저장된 영상을 날짜별로 분류하여 저장한다. 날짜 옆 "[]" 안에 숫자는 그 날짜의 저장된 파일의 개수이다.

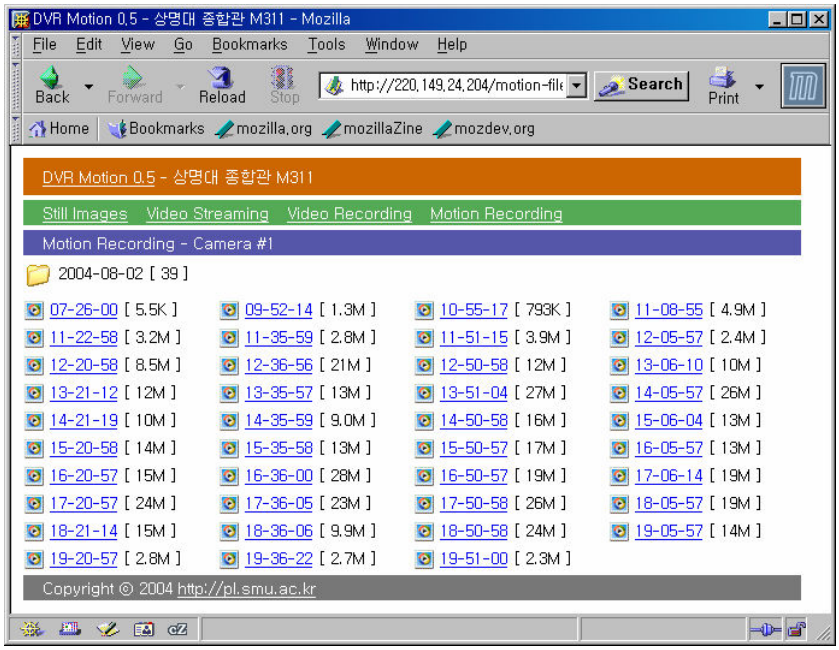
3.5 전체 저장된 영상 시간대별 목록



<그림 6.24> 전체 저장된 영상 시간대별 목록

그림 6.24는 전체 저장된 영상을 시간대별로 보여준다. DVR Motion 웹 페이지의 "Video Recording" 메뉴를 클릭한 후 날짜별 목록에서 원하는 날짜를 클릭하면 정해진 시간대별로 저장된 mpeg 파일 목록을 볼 수 있다. 시간대별 옆 "[]" 안에 크기는 파일 용량이다.

3.6 모션만 저장된 영상 시간대별 목록



<그림 6.25> 모션만 저장된 영상 시간대별 목록

그림 6.25는 모션만 저장된 영상을 시간대별로 보여준다. DVR Motion 웹 페이지의 "Motion Recording" 메뉴를 클릭한 후 날짜별 목록에서 원하는 날짜를 클릭하면 정해진 시간대별로 저장된 mpeg 파일 목록을 볼 수 있다.

참고문서

[1] 4 Front Technologies, http://www.4front-tech.com/

[2] Advanced Linux Sound Architecture, http://www.alsa-project.org/

[3] Alan Cox, Video4Linux API Kernel Documents, http://pl.smu.ac.kr/researches/projects/linux-dvr/v4l-kernel-doc

[4] Alex Buell, "Framebuffer HowTo",http://www.tldp.org/HOWTO/Framebuffer-HOWTO.html, 2000.

[5] ALSA(Advanced Linux Sound Architecture), http://www.alsa-project.org

[6] ALSA Sound Card Matrix & Install documentation, http://www.alsa-project.org/alsa-doc

[7] Ankur Mahajan and Dongha Shin, "Simple Image Grabber", http://pl.smu.ac.kr/lxr/simple-image-grabbler/source/, 2003.

[8] Audio File Formats FAG, http://sox.sourceforge.net/AudioFormats.txt

[9] Ben Salama, "UNIX System Programming 2/e", Addison-Wesley, 1999.

[10] BTTV, http://bytesex.org/bttv

[11] Eric Sandeen, The BTTV Mini-HOWTO v0.3 (http://metalab.unc.edu/LDP), February 2000.

[12] ffmpeg Multimedia System, http://ffmpeg.sourceforge.net

[13] GLUT - OpenGL Utility Toolkit, http://www.opengl.org/resources/libraries/glut.html

[14] holelee, "Framebuffer 이야기", http://kelp.or.kr/korweblog/search.php?query=frame+ buffer+ 이야기, 2002.

[15] http 1.0 protocol, http://www.w3.org/Protocols/rfc1945/rfc1945

[16] Installing Open Sound System, http://www.4front-tech.com/install_gzipped.html

[17] Jaroslav Kysela, Abramo Bagnara, Takashi Iwai, and Frank van de Pol, ALSA Library API on-line documentation, ALSA Project, http://www.alsa-project.org/alsa-doc/alsa-lib/, 2001.

[18] Jeff Tranter, Open Sound System Programmer's Guide, 4 Front Technologies, http://www.opensound.com/pguide/oss.pdf, 2000.

[19] KLDP wiki, X 윈도우 프로그래밍 기초과정, http://www.kldp.orgwiki.php/LinuxdocSgml/X-Window-Programming-KLDP

[20] LessTif, "LessTif Documentation", http://www.lesstif.org/Lessdox/lesstif.html

[21] Linux Kernel, http://www.kernel.org

[22] Linux OVCam Drivers, http://alpha.dyndns.org/ov511

[23] Linux Support for Philips USB webcams, http://www.smcc.demon.nl/webcam

[24] Logitech QuickCam Zoom,http://www.pcuf.fi/~teva/Linux/webcam/webcam_part0.html

[25] man page와 info page.

[26] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner, "OpenGL Programming

Guide, Third Edition", Addison Wesley, 1999.
[27] Maxwell Sayles, "How To" for Video For Linux (VFL), http://pages.cpsc.ucalgary.ca/~sayles/VFL_HowTo/, 2002.
[28] Motif Programming Manual, http://www.ist.co.uk/motif/download/6A/6A_book.pdf
[29] Motif Reference Manual, http://www.ist.co.uk/motif/download/6B/6B_book.pdf
[30] Motion, http://motion.sourceforge.net
[31] MPEG-2 FAG, http://www.cise.ufl.edu/help/software/doc/mpeg/MPEG-FAQ
[32] Naoyuki Ichimura, "Image Capture by Video4Linux", http://staff.aist.go.jp/naoyuki.ichimura/research/tips/v4ln_e.htm
[33] OpenGL, http://www.opengl.org
[34] OSS(Open Sound System), http://www.opensound.com
[35] Open Sound System/Free, http://www.opensound.com/ossfree/index.html
[36] Open Sound System Programming's Guide, http://www.opensound.com/pguide/oss.pdf
[37] Paul Bourke, PPM / PGM / PBM image files, http://astronomy.swin.edu.au/~pbourke/dataformats/ppm/, 1997.
[38] Robert T. Collins, Alan J. Lipton, Takeo Kanade, Hironobu Fujiyoshi, David Duggins, Yanghai Tsin, David Tolliver, Nobuyoshi Enomoto, Osamu Hasegawa, Peter Burt and Lambert Wixson, "A System for Video Surveillance and Monitoring: VSAM Final Report," Technical report CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, May, 2000.
[39] SuSE, http://www.suse.com
[40] Takashi Iwai, NOTES ON KERNEL OSS-EMULATION, ALSA Project, http://www.alsa-project.org/~iwai/OSS-Emulation.html, 2003.
[41] The Mesa 3D Graphics Library, http://www.mesa3d.org
[42] v4l kernel Doc API, http://www.unix-systems.org/single_unix_specification
[43] UNIX SYSTEM V RELEASE 4 Programmer's Guide: XWIN Graphical Windowing System Xlib-C Language Interface, AT&T, UNIX Software Operation
[44] VFL How to, http://pages.cpsc.ucalgary.ca/~sayles/VFL_HowTo
[45] Video Codecs and Pixel Formats, www.fourcc.org
[46] Video for Linux Two - Image Data Formats, http://www.thedirks.org/v4l2/v4l2fmt.htm
[47] video4linux resources, http://www.exploits.org/v4l
[48] W. Richard Stevens, "UNIX Network Programing Volume2(Interprocess Communications)", Prentice Hall, 2002.
[49] W. Richard Stevens, "Advanced Programming in the UNIX Environment", ADDISON-WESLEY, 1992.
[50] xawtv homepage, http://linux.bytesex.org/xawtv
[51] XFree86, http://www.xfree86.org
[52] XFree86, "Documentation and Resources", http://xfree86.mirror.or.kr/4.4.0

[53] XFree86 Manual page, http://xfree86.org/4.4.0/manindex3.html
[54] XVideo 예제 프로그램, testxv.c, http://bellet.info/XVideo/testxv.c
[55] 권태완, Digital Image Processing, http://mcl.hallym.ac.kr/twkwon/2_lecture/Digital_Image_Processing, 2003.
[56] 김보람, OSS 오디오 프로그래밍, http://project.oss.or.kr/linuxdvr
[57] 김충석, "X 윈도우 시스템 프로그래밍", 이한출판사, 1994.
[58] 서영진, Linux Multimedia Programming, http://user.chollian.net/~valentis/Suhdang/QT_Programming/Lecture_MM.html, 2003.
[59] 신동하, 이우철, Video4Linux를 이용한 모션 검출 알고리즘의 구현, "소프트웨어 미디어 연구소", 상명대학교, 2004.
[60] 이만재, 홍명희, 박현재, "멀티미디어 개론", 한국방송대학교출판부, 1999.
[61] 이필두, "디지틀 영상 이론에서 활용까지-Premiere & After Effects",(주)영진닷컴, 2002.
[62] 이국현 역, "Vesafb mini HowTo", http://wiki.kldp.org/wiki.php/LinuxdocSgml/Vesafb, 1999.