다중 카메라 응용을 위한 카메라 프레임워크

이혜민 O, 신동군

성균관대학교 전기전자컴퓨터공학과

gudbooy@skku.edu, dongkun@skku.edu

Camera Framework for Multiple Concurrent Camera Applications

Hyemin Lee O, Dongkun Shin

Department of ECE, Sungkyunkwan University

요 약

센서 장치 중 사물인터넷 응용에 가장 널리 쓰이는 센서 중 하나는 카메라다. 카메라를 통해 얻은 영상 데이터는 비전 프로세싱을 통해서 시멘틱 정보로 변화되어 사용자에게 제공되거나, 녹화나 사진 촬영을 통해 사용자에게 다양한 서비스를 제공할 수 있다. 따라서, 카메라 관련 응용이 동시에 실행되며 사용자에게 다중 서비스를 제공해야 하는 경우가 증가하고 있다. 하지만, 기존 운영체제 시스템에서 다중 카메라 응용을 동시에 운용할 때 카메라를 선점한 프로세스 외 다른 프로세스들은 카메라 장치에 대한 기아를 겪는다. 이러한 문제는 결론적으로 카메라 응용과 서비스의 확장성을 떨어뜨리는 결과를 초래한다. 이를 해결하기 위해 *카메라 공유 관리자*(*Camera Sharing Manager*)라는 데몬 프로세스를 배치하여 여러 응용에서 발생시킨 요청을 처리하고 카메라 프레임 버퍼를 공유 메모리 영역에 할당함으로써 다중 카메라 응용의 동시 운용을 지원한다. 실험 결과로는 4개, 8개의 FullHD(1080P)영상을 동시 녹화 시 30FPS(Frame Per Seconds) 기준 대비 각각 평균 1.85, 4.5FPS의 감소를 한다.

1. 서 론

최근 사물인터넷 관련 기기의 폭발적인 증가로 인해 다양한 센서를 기반으로 사용자에게 여러가지 서비스를 제공하는 응용들이 등장하고 있다. 특히, 카메라를 활용한 다양한 응용들이 생김으로써 여러 카메라 관련 응용을 동시에 운용해야 하는 요구가 늘어난다. 예를 들어, 차량용 블랙박스의 경우 영상 녹화와 더불어 ADAS(Advanced Driver Assistance System)와 같은 비전 프로세싱의 동시 운용이 필히 요구된다. 하지만, 기존의 운영체제(예. 리눅스)에서는 카메라 장치에 대해 단일 응용의 배타적인 접근만 허용한다[1, 2]. 이러한 문제는 단순히 녹화, 사진촬영에 국한된 것이 아닌 사진이나 영상을 분석하여 사용자에게 고수준의 시멘틱을 제공하는 비전 라이브러리(OpenCV)에서도 발생한다[1]. 이는 궁극적으로 사물인터넷 플랫폼에서 여러 카메라 응용을 동시 운용할 때 확장성을 떨어뜨린다.

카메라 장치의 프로세스 간 기아 현상은 사물인터넷에서 흔히 사용되는 자바스크립트 기반 플랫폼[3, 4, 5]에서 더욱 악화된다. 자바스크립트 기반 플랫폼의 비동기 입출력 방식 때문에 프로세스 간 기아 현상은 비동기 입출력 함수 간 기아현상으로 확대된다. Node.js[3]는 비동기 입출력을 처리할 때 자바스크립트 실행 스레드(메인 스레드)의 블럭을 방지하기 위해 입출력 마다 각각의 스레드를 생성하여 처리한다. 그림1(a)에서 비동기 입출력 방식에서 발생하는 기아를 묘사한다. 메인 스레드에서 두 개의 카메라 입출력 요청(A, B)이 중첩되는 경우 먼저 호출된 요청 A의 스레드는 카메라의 접근이 용인되지만, 요청 B의 스레드에서는 카메라 장치 접근이 거절된다. 따라서, 사물인터넷에서 흔히 사용되는 비동기 입출력 방식의 응용 내에서 카메라 장치에 대한 기아 문제는 반드시 해결되어야 한다. 따라서, 본 논문은 비전 라이브러리, 녹화, 사진촬영, 카메라 네트워크 스트리밍과 같은 여러 카메라 응용을 동시에 운용하고 비동기 입출력 방식의 Node.js와 같은 사물인터넷 플랫폼을 위한 확장성 있는 카메라 프레임워크를 제시한다.

2. 본 론

2.1 Video For Linux 2(V4L2)의 확장성 문제

리눅스는 카메라, 코덱, 비디오 장치들을 관리하거나 통제하기 위한 리눅스 미디어 인프라인 V4L2(Video For Linux 2)라는 API를 제공한다. 카메라의 프레임 버퍼는 V4L2에서 제공하는 3가지의 스트리밍 입출력 API를 통해 사용자 영역으로 제공된다[6]. 첫 번째는, *Read* 시스템 콜을 통해 프레임 버퍼를 획득한다. 두 번째, *User Pointers* 기법으로 프로세스의 가상 메모리 공간에 프레임 버퍼를 직접 제공하는 기법이며, 마지막은 *메모리 맵핑*으로 mmap() 시스템 콜을 통해 프로세스의 가상 메모리 공간과 커널 영역에 카메라 버퍼 영역을 맵핑하여 제공하는 기법이다. V4L2의 카메라 버퍼 제공 기법은 모두 하나의 프로세스 내 가상 메모리 공간으로 국한 되어있으며, 만약 특정 프로세스가 카메라 디바이스를 선점하는 경우 다른 프로세스는 카메라의 접근이 제한된다. 이러한 문제를 해결하기 위해 카메라 관련

이 논문은 2013년 정부(미래창조과학부)의 재원으로 (재)스마트IT융합시스템 연구단(글로벌프론티어사업)의 지원을 받아 수행된 연구임((재)스마트IT융합시스템 연구단-2011-0031863)

응용들의 요청을 통합적으로 처리하며, V4L2를 직접 호출하여 버퍼를 공유 메모리 영역에 적재하는 형태의 *카메라 공유 관리자(Camera Sharing Manager)*를 제안한다.

2.2 공유 메모리 기법

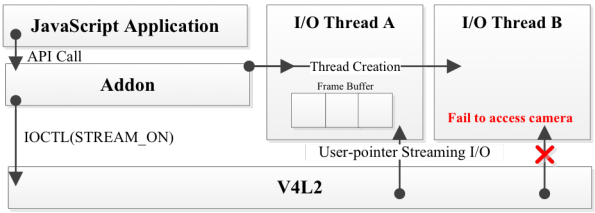
하나의 프로세스에만 제공되는 카메라 프레임 버퍼를 효율적으로 공유하기 위해 System V IPC(프로세스 간 통신)방법 중 하나인 공유 메모리를 사용해 프레임 버퍼를 공유한다. 앞서 언급했듯이 V4L2는 3가지의 스트리밍 입출력을 지원한다. read() 시스템 콜을 통한 입출력 방식과 mmap() 시스템 콜을 통한 스트리밍 방식은 공유 메모리로 프레임 버퍼를 할당 시 한 번의 부가적인 메모리 복사가 불가피하다. 이는 높은 해상도와 비트레이트를 가진 카메라 프레임을 적재하는 경우 FPS(Frame Per Seconds)를 떨어뜨리는 결과를 초래한다. 따라서, 부가적인 메모리 복사 없이 공유 메모리로 카메라 프레임을 제공할 수 있는 User Pointers 스트리밍 입출력 방식을 통해 V4L2에서 직접 공유 메모리 영역에 프레임을 할당하여 추가적인 메모리 복사 작업 없이 프레임을 공유 메모리 공간으로 적재한다.

2.3 프레임 버퍼 동기화

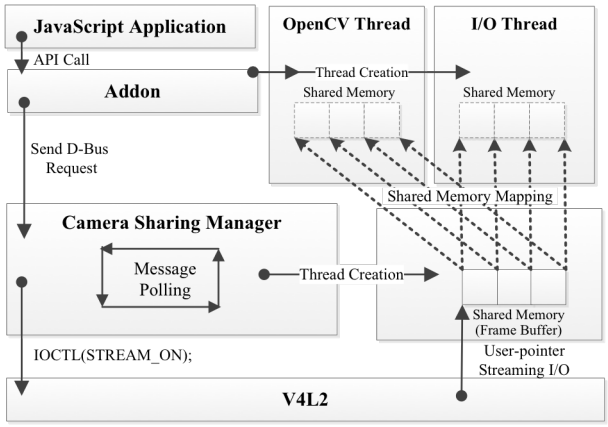
여러 프로세스 간 카메라 프레임 버퍼 동기화를 위해 다음의 문제를 고려한다.

첫 번째, 다중 소비자를 위한 카메라 프레임 버퍼 획득에 대한 시간 동기화가 필요하다. 이에 대한 배경설명으로 V4L2에서는 카메라 프레임의 시간 동기화를 위해 두 가지의 입출력 Multiplexing API(select(), poll() 시스템 콜)를 제공한다. 해당 시스템 콜이 호출되면 카메라 장치의 파일 서술자 변화(카메라 프레임의 도착 여부)를 감지하여 프레임 간 시간 동기화를 제공한다. 따라서, *카메라 공유 관리자*(생산자)는 V4L2에서 제공하는 Multiplexing API를 통해 카메라 프레임을 정확한 시간에 맞추어 공유 메모리에 적재할 수 있지만, 해당 프레임의 소비자는 V4L2에서 제공하는 Multiplexing API를 사용할 수 없다. 왜냐하면, 비결정적인 프로세스 스케줄링에 따라 Multiplexing API에서 파일 서술자의 변화를 감지하지 못할 가능성이 존재한다. 두 번째, 공유 메모리 영역에 대한 입출력이 발생하면서 생기는 버퍼 훼손을 방지할 수 있는 동기화 기법이 필요하다. 예를 들어, 공유 메모리 내 프레임을 적재하는 도중 소비자가 해당 영역에 접근 시 소비자는 훼손된 프레임을 가지게 된다. 세 번째는 소비자 프로세스들과의 입출력 동기화 메커니즘으로 인한 *카메라 공유 관리자*의 FPS 감소를 최소화해야 한다. 모든 소비자는 *카메라 공유 관리자*로부터 프레임 버퍼를 받기 때문에 *카메라 공유 관리자*의 지연은 즉 모든 소비자의 지연으로 확대된다.

세 가지 문제를 해결하는 방법은 다음과 같다. 첫 번째, 소비자는 Multiplexing 입출력 대신, 루프를 돌며 프레임을 획득한다. 이로 인해 발생하는 중복 프레임 획득 문제를 해결하기 위해 각각의 프레임 버퍼의 타임 시퀀스와 페이로드 사이즈를 확인하여 중복된 프레임 버퍼의 획득을 방지한다. 두 번째, 공유 메모리 영역에 대한 입출력 동기화는 POSIX 세마포어를 이용해 입출력 동기화 프로토콜을 구성하여 공유



(a) Asynchronous I/O Camera-related applications issue



(b) Camera Sharing Manager request managing operation

그림 1. 카메라 공유 관리자의 요청 관리 프로세스

버퍼에 대한 입출력 동기화를 제공한다. 세 번째, 소비자와 생산자 간 입출력 동기화 오버헤드에 의해 발생한 생산자의 FPS 감소를 방지하기 위한 기법으로 두 개의 공유 버퍼를 할당하여 RCU(Read-Copy-Update) 형태의 이중 버퍼링을 제공한다. 예를 들어 *카메라 공유 관리자*가 새로운 프레임을 공유 버퍼 A에 적재할 때 다른 소비자들이 버퍼 A에 대한 복사(비전 프로세싱)나 파일 입출력(녹화) 작업을 끝내지 않았을 경우 생산자는 A 버퍼가 아닌 A' 버퍼에 새로운 프레임을 적재하여 소비자의 지연과 관계없이 생산자는 카메라 프레임을 정확한 시간 간격 내 획득할 수 있다.

2.4 어플리케이션 요청 관리 및 API

*카메라 공유 관리자*는 시스템 내 데몬 프로세스로 운영된다. *카메라 공유 관리자*는 각각의 응용들이 발생시킨 요청들을 통합하여 적절하게 서비스를 중재해야 하고 요청에 맞추어 V4L2 API를 통해 카메라 장치에 명령해야 한다. 그림1(b)는 카메라 공유 관리자의 요청 처리 프로세스를 묘사한다. 카메라 공유 관리자는 응용과의 통신을 위해 프로세스 간 통신 메커니즘(IPC) 중 리눅스, 타이젠에서 사용되는 메커니즘인 D-Bus를 사용한다. *카메라 공유 관리자*는 메인 스레드에서 D-Bus 메시지의 여부를 확인하며, 요청 발생 시 카메라 프레임을 공유 버퍼에 적재하는 스레드를 동적으로 생성하여 메모리 복사 없이 여러 응용에 제공한다.

카메라 관련 자바스크립트 API는 Node.js에서 제공하는 기능인 *Addon*을 기반으로 구현한다. *Addon*은 C/C++로 작성된 공유 오브젝트와 자바스크립트 API간 인터페이스를 제공하는 기능이다. 카메라 관련 자바스크립트 API를 호출 시 *Addon* 내에서 입출력 스레드를 동적으로 생성하여 공유 메모리 영역에 접근하여 프레임 버퍼를 처리한다. 처리방식에

따라 녹화, 사진촬영, 네트워크 스트리밍과 같은 기능 등을 구현한다. OpenCV 라이브러리는 추상화된 카메라 입출력 인터페이스의 구현을 카메라 공유 관리자와 통신하도록 구현했다. 이를 통해 기존 OpenCV 응용의 코드 수정 없이 카메라 공유 관리자와 통신을 할 수 있다. 따라서, OpenCV 라이브러리는 Addon으로 추가하여 자바스크립트 API 형태로 사용할 수 있을 뿐만 아니라, C/C++ 어플리케이션을 통해 사용할 수 있다.

3. 실험과 분석

3.1 실험 환경

실험은 Raspberry-Pi2(CPU : BCM2836 900Mhz ARM Cortex-A7 Quad Core, SDRAM : 1GB)를 사용하였으며, 카메라 모듈은 MIPI Interface기반의 OV5647이다. BCM2836 SoC에서 지원하는 비디오 코덱은 MJPEG, H.264을 이며, 성능은 FullHD(1080P) 30FPS를 보장한다. 운영체제는 Raspbian(Linux Kernel 4.1.13) 이다.

3.2 제한 사항

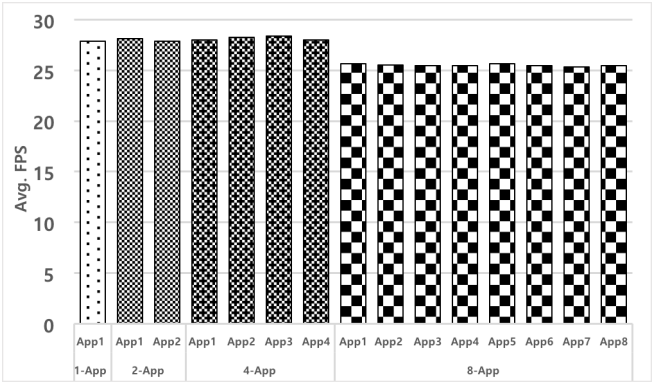
BCM2836 SoC에서 제공하는 비디오 코덱의 한계로 인해 동시에 여러 색 영역(Pixel Format)에 대한 접근이 제한된다. 예를 들어, 영상 녹화(MJPEG, H.264)와 OpenCV(RGB, YUV420)의 동시 운용에 제한이 있다. 이를 최대한 극복하기 위해 *카메라 공유 관리자*는 정적으로 특정 픽셀 포맷에 대한 우선순위를 정해 선점형 스케줄링을 하여 서비스를 제공한다. 이러한 제한은 멀티 포맷 하드웨어[8]로 제거할 수 있다.

3.3 실험 결과

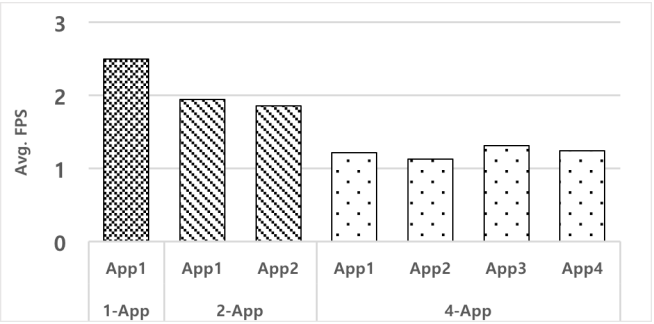
두 가지의 확장성 관련 성능 분석 실험을 진행하였다. 그림 2(a) 실험은 FullHD(1080P)영상에 대한 동시 녹화 시 평균 FPS이다. 1, 2, 4, 8개의 녹화를 동시 운용할 때 각각 평균 27.88, 28.00, 28.15, 25.5FPS이다. *카메라 공유 관리자*의 FPS는 평균 28.61FPS로 2.3절(다)에 제시한 기법으로 카메라 공유 관리자의 FPS 감소가 상대적으로 최소화 되었다. 동시 녹화 시 1.8~4.5FPS 내 감소율은 영상의 문제를 일으키지 않으며, 심지어 8개의 동시 녹화에서도 30FPS 영상과의 뚜렷한 차이가 없다. 두 번째 실험은 480P(640*480)카메라 영상으로 1, 2, 4개의 얼굴 인식 비전 어플리케이션을 수행했을 때의 평균 FPS는 각각 2.50, 1.90, 1.23FPS로 영상 녹화와 다른 양상을 보인다. 이는 비전 응용들의 특징으로 프레임 버퍼를 매트릭스로 복사하는 추가 과정이 발생하며, 압축 포맷이 아닌 RGB나 YUV420과 같은 Raw 색 영역을 사용하여 메모리 오버헤드가 영상에 비해 월등히 높다. 더욱이, 연산에 필요한 CPU 작업량이 녹화나 사진촬영 네트워크 스트리밍에 비해 매우 높다. 따라서, 이러한 문제점을 해결하기 위해 [1]에서 제안한 비전 라이브러리에서 제공하는 함수에 대한 연산 작업 메모이제이션 기법이나, 고성능 서버로 연산 작업량이 많은 작업을 오프로딩하는 등에 기법은 향후 연구로 진행하겠다.

4. 결 론

사물인터넷 장치에서 카메라 자원의 확장성 지원은 다른 센서에 비해 매우 중요하다. 따라서, 이 논문에서는 여러 카메라 관련 응용을 동시에 운용하기 위한 기법으로 *카메라 공유 관리자*라는 데몬 프로세스를 배치하여 프레임 버퍼를 공유하고 이에 대한 정확한 동기화를 제공하는 카메라 프레임워크를 제시한다. 이를 통해 응용들에 대한 카메라 확장성을 실험을 통해 증명했다.



(a) Concurrent Recording - MJPEG 1080P



(b) Concurrent Vision Processing - Face Detection 480P

그림 2. 카메라 확장성 실험 결과

참 고 문 헌

[1] LiKamWa, Robert, and Lin Zhong, "Starfish: Efficient Concurrency Support for Computer Vision Application," *Proceeding of the 13th Annual International Conference on Mobile Systems, Applications, and Services*., ACM, pp. 213-226, 2015.

[2] Amiri Sani, Ardalan, et al., "I/O Paravirtualization at the device file boundary," *Proceedings of the 19th international conference on Architecture Support for Programming Language and Operating System,* ACM, pp. 319-332, 2014.

[4] Tilkov, Stefan, and Steve Vinoski, "Node.js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, 14.6, 2010.

[5] "Mongoose IoT Platform: IoT Software Platform.," [Online]. Available: https://www.cesanta.com (downloaded 2016. May. 1)

[6] "IoT.js: A framework for Internet of Things.," [Online]. Available: https://samsung.github.io/iotjs (downloaded 2016. May. 1)

[7] "V4L2: LINUX MEDIA INFRASTRUCTURE API.," [Online]. Available: http://linuxtv.org/downloads/v4l-dvb-apis/ (downloaded 2015. Aug. 15)

[8] "MFC : Samsung Multi Format Codec.," [Online]. Available : http://linux-exynos.org/wiki/MFC (downloaded 2016. May, 1)