제 18 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

RTLinux 기반 영상처리 모듈 개발

최종황, 문승빈
세종대학교 컴퓨터 공학과
e-mail : sbmoon@sejong.ac.kr

RTLinux for image processing module development

Jonghwang Choi, Seungbin Moon
Dept. of Computer Egineering, Sejong University

요 약

실시간 제어 시스템은 그 특수성에 의해 개발자에게 상당한 제약이 있었다. 본 논문에서는 실시간 운영체제 중에서 Linux 를 기반으로 한 RTLinux 를 사용하였다. RTLinux 를 StrongARM SA-1110 CPU 가 탑재된 영상처리 보드에 올리는 과정을 기술하고, RTLinux 의 module 이라는 작업 형태를 이용하여 로봇의 카메라에서 획득한 영상을 실시간으로 전송하도록 구현하였다. 또한, 사용자의 입력을 감시하는 부분을 실시간 모듈로 구현하여 제한된 시간 안에 원하는 결과를 산출하도록 구현하였다.

1.서 론

RTLinux 는 실시간 작업을 다루기 위한 Linux 의 확장 선상에 있는 제어용 컴퓨터 운영 체제 이다. 예를 들어, Data Acquisition, Control, Robotics 등과 같이 주어진 시간 안에 원하는 결과를 도출해야 하는 system 이 필요로 하는 분야에 사용되어 진다. RTLinux 는 Linux Kernel 에 작은 Hard Real Time Kernel 을 추가하여 이루어 진다.[1]

일반적인 운영체제에서는 선점형 멀티태스킹 시스템이다. 이러한 체제에서 정확한 시간주기를 가지고 일을 해야 하는 프로세서에게 커널이 정확한 타이밍을 맞춰준다는 것은 불가능 하다. 일반적인 운영체제는 기껏해야 밀리초 단위의 정확성으로 제어를 하지만, 통신기기나 정밀 제어의 경우 수십 마이크로초 단위로 정확하게 시간을 측정해야 하는 경우가 많다. 다시 말하자면 인터럽트가 발생하였을 때 해당 프로세스나 태스크가 수십 마이크로초 이내로 동작해야 하는 경우에는 일반적인 범용 운영체제로는 이러한 조건을 만족 할 수가 없다. 이러한 점을 만족 시키기 위해 실시간 처리가 필요하게 되었다.[2]

기존에 실시간 제어를 위해서는 상용 실시간 OS 를 구입하여 사용하였다. 이러한 운영체제에는 VxWorks, VMEexec, pSOSystem, uCOS Real-Time Kernel 등이 사용되어 오고 있다. 그러나 별도의 개발 kit 와 Runtime License 를 구입해야 하므로 비교적 비용이 많이 드는 방법이다. 실시간 시스템은 그 특성상 해당 시스템에 맞게 설계/제작된 보드에 내장되고, 소스가 공개되지 않는다. 이러한 제약을 피하기 위해 본 논문에서는 개발자가 커널의 소스를 직접 수정 할 수 있는 Linux 를 사용한 RTLinux 를 이용하였다. 유닉스 계열 운영체제는 시분할 스케쥴링 방식을 사용하도록 고안되었으며 리눅스 스케쥴러는 프로세서를 공평하게 공유할 수 있도록 만들어 졌다. 그러나 Robot 제어와 같은 hard real time 을 만족하기 어려운 실정이다. 이러한 실시간 시스템 운영 체제를 직접 구현하는 것보다는 기존의 유닉스 계열의 운영체제를 실시간 시스템에 사용할 경우 많은 장점이 존재한다. 특히 리눅스와 같이 공개된 운영 체제를 사용하여 새로운 실시간 운영체제의 기반이 아닌 기존 운영체제의 확장으로 실시간 시스템을 구성한다면, 개발자들에게 익숙한 개발환경을 제공함으로써 개발 시간을 단축할 수 있다. [3-4]

특히 로봇 시스템에서 로봇의 동작과 제어는 반드시 실시간 제어가 가능한 시스템이어야 한다. 오 동작이 발생할 경우 작업 시스템의 손실과 함께, 생산률의 저하로 이어지고, 작업환경내의 대인피해가 발생 할 수 있기 때문이다.

로봇에 비전 시스템을 적용한다는 것은 비록 제한적이지만 로봇 스스로 주위 상황을 판단하여 적절한 대응을 할 수 있도록 하기 위한 것으로서 로봇에서 일정한 지능을 부여하기 위한 핵심적인 수단이다.

본 논문에서는 RT-Linux 를 ARM board 에 탑재하여 로봇의 제어와 카메라에서 획득한 영상을 처리하는 시스템을 구현을 하였다.

본 논문의 구성은 다음과 같다. 2 절에서는 개발환경에 있어서의 하드웨어 구성과 각 기능을 설명하고 제

어 할 로봇을 소개한다. 3 절에서는 RTLinux 의 동작 원리와 구성에 대해 설명하고, ARM board 에 올라가는 모듈과 그들의 상호동작에 대해 기술한다. 4 절에서는 영상획득 과정과 RTLinux 의 처리과정을 기술한다. 5 절에서는 RT module 간의 상호 메커니즘을 기술한다.

2.하드웨어 구성

하드웨어 구성은 그림 1 은 내장형 장비의 실물인 RABBIT board 의 사진이다. 보드에 RTLinux 를 탑재하기 위해 cross cable 과 NFS 파일 시스템 및 JTAG 을 이용한다. 그림 1 에서 보듯이 사용자의 입출력을 받을 수 있는 방법은 serial mouse 로만 가능하다. RTLinux porting 과 어플리케이션 수행을 위해서는 키보드의 입력을 받아야 한다. 그 부분은 RS232C 와 Cross cable 로 구성하였다. 그림 2 에서는 하드웨어 구성도를 나타내고 있다. 그림 2 에서 볼 수 있듯이, CPU 는 StrongARM SA-1110 32bit RISC 를 이용하였고, 메모리는 SDRAM 32MB, Flash Memory 16MB 를 이용하였으며, 호스트 장비와의 통신을 위해 RS232C 를 사용하였다. 이 모델은 Assabet 를 기반으로 만들어 졌다.[5]

해당 영상을 CRT 에 디스플레이 하기 위해서 MQ200 Graphic chipset 을 사용하였다. 이 제품은 저전력, 고성능을 그 장점으로 하고 있으며, 2D Graphic Accelerator 기능을 가지고 있다.[6]

ARM board 와 연결된 로봇 AT2 는 그림 3 에서 보여지고 있으며, ARM board 와는 serial 로 연결되어 있다. 로봇의 제일 말단부에 카메라를 부착하여 영상을 획득하는 방법을 사용하였다. 카메라에서 잡은 영상은 로봇 시스템에 전달되고, 다시 serial 케이블로 전송하는 방법을 채택하였다. 전송 프로토콜은 protocol3 을 사용한다.

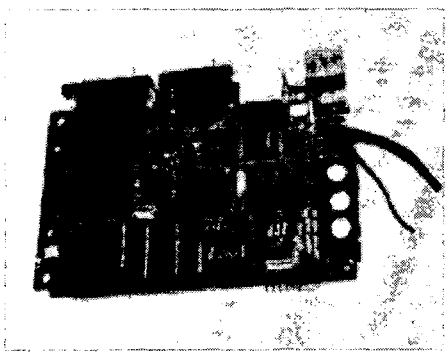

그림 1. RABBIT Board

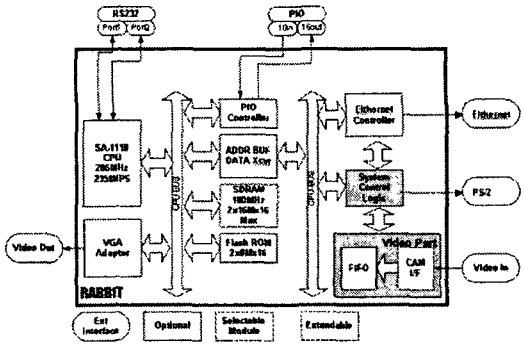

그림 2. Schematic Diagram

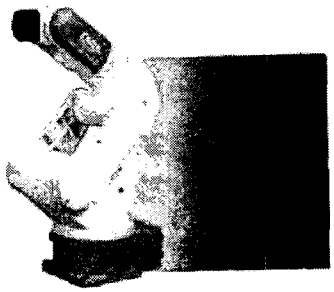

그림 3. Robot

3. RTLinux 와 처리 module

3.1 RTLinux

일반적으로 Linux 는 hard real-time 을 요구하는 시스템에서는 적합하지 않다. 이는 Linux 커널이 hard real-time 응용 프로그램에 대한 지원을 고려해서 설계된 시스템이 아니라, 일반적인 desktop 응용 프로그램을 수행할 목적으로 time-sharing 에 기반하기 때문이다. 즉, 시스템을 사용하는데 있어서 균등한 배분만을 고려했지, hard-real time 시스템과 같은 deadline 이 앞선 우선순위가 높은 task(or process)로의 kernel 내에서 preemption 을 지원하지 않기 때문이다.

이러한 리눅스가 가지는 특성을 RTLinux 에서는 우회하는 방법을 사용한다. 즉, 커널이 hard real-time process 를 직접적으로 지원하는 것은 Linux 커널 전체를 새로이 디자인하는 만큼의 노력이 들기 때문에, Linux kernel 을 하나의 real-time task 로 보고, 다른 real-time task 가 없을 경우에만 Linux 를 수행하도록 만드는 방법을 사용하고 있는 것이다. 이것은 실질적으로 하나의 컴퓨터 시스템에 두개의 커널을 동작시키는 방법으로 hard real-time task 에 우선순위를 둔다는 것이다. 이렇게 함으로서 Linux 커널을 수정해 주어야 하는 노력은 최소화 하고, 동시에

hard real-time 을 구현할 수 있는 방법을 제시하는 것이 RTLinux 가 생각한 방법이다. 즉, 응급한 hard real-time 특성을 가지는 프로그램을 작게 유지하고, 나머지 데이터에 대한 복잡한 처리는 특별한 인터페이스를 통해서 Linux task 로 전달되어 처리되도록 만든 것이다. 이때 사용하는 것이 그림 5 의 real-time FIFO 이다.[7]

RTLinux 의 내부 동작 원리는 그림 4 와 같다. 그림 4 에서 보시다시피 Linux kernel 을 RT-task 와 같이 하나의 Real Time task 로 취급하고 있다.

로봇의 동작과 이미지를 받는 부분은 일반 리눅스에서 task 를 하나 생성하여 모니터에 보여준다.

그림 5 의 RTLinux 의 FIFO 는 Linux process 와 통신할 수 있는 창구 역할을 하는 메커니즘이다. Linux 에서 말하는 FIFO 와 거의 동일한 개념이지만, 일종의 device node 의 형태로 구현해서 Linux process 에서는 단지 이 device file 을 open 해서 읽고/쓰기를 수행할 수만 있다. RTLinux 의 thread 는 이러한 FIFO 에 자신이 획득한 정보를 기입하면, 나중에 Linux process 에서 이를 읽어간다. 모든 external/internal 한 이벤트는 먼저 RTLinux 의 관심이 되며, Linux process 들은 사용자 영역에서 그리고, RTLinux thread 들은 커널영역에서 수행되기에 이들간의 연결고리 역할을 하는 것이 바로 RTLinux 의 FIFO 라는 것을 알 수 있다.

RT-task1 은 user 의 input 을 받아들이고, RT-task2 는 로봇과 시리얼 통신을 담당하는 부분으로 RT-FIFO 가 경로역할을 하게 된다.[8]

모든 RT-task 는 모듈로써 동적으로 loading 되어 커널과 link 된다.

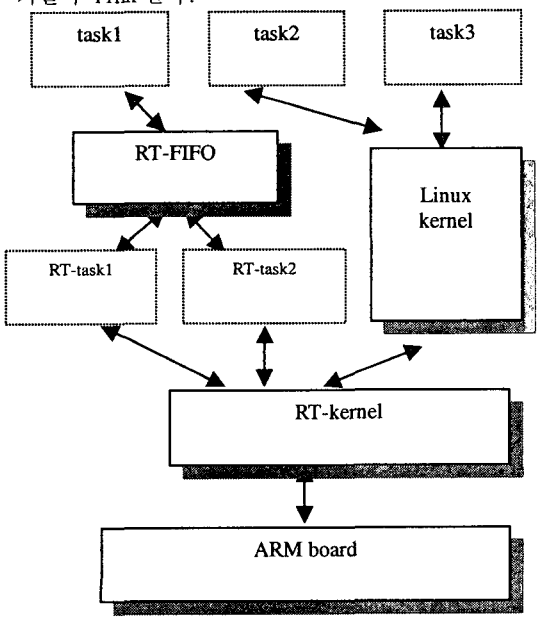

그림 4 RTLinux 동작원리

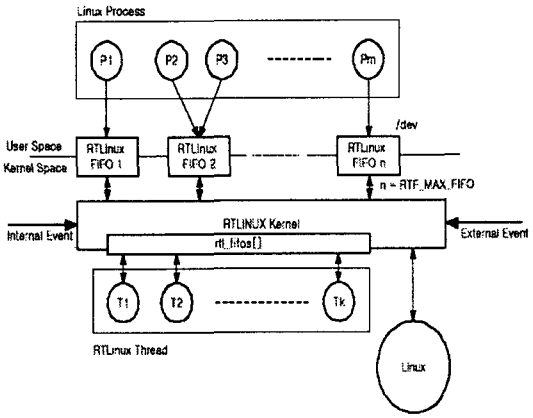

그림 5 RTLinux FIFO 의 구조

3.2 RTLinux porting

RTLinux 를 ARM board 에 올리는 작업은 다음과 같다. 일반 Linux kernel version 2.4 이상이어야 한다.

이러한 kernel 을 ARM board 에 올리려면 해당 ARM CPU 에 맞게 patch 를 수행하여야 하고, board 에 맞게 patch, 마지막으로 RTLinux 과 동작하기 위하여 RT patch 를 수행한다. RTLinux 는 version3.0 을 arm 보드에 맞게 patch 를 한 후에 위의 커널을 링크 시키면 RTLinux 커널이 형성된다.

생성된 커널을 JTAG 을 이용해 해당 ARM board 의 flash 메모리에 ramimage 와 같이 올린다. 기타 device module 과 RT module 은 NFS(Network File System)을 이용하여 host PC 에서 ARM board 로 옮겨져서 수행하게 된다.

RTLinux 는 모듈로써 동작하게 되어있다. 모듈 올리는 순서는 rtl.o, mbuff.o, rtl_time.o, rtl_posixio.o, rtl-fifo.o 순이다.

RTLinux 에서 program 을 돌리기 위해서는 CPU 에 맞게 Cross Compiler 을 하게 되는데 linux-arm-gcc 를 사용한다. GUI 환경은 Embedded-QT-2.3.0 을 사용하여 구현하였다.

4. 영상처리

영상처리는 크게 2 가지 부분으로 나눌 수 있다. 입력 영상을 받기 시작하면서 로봇으로부터 들어오는 영상을 받는 task 와 받은 영상을 처리하는 부분이다.

두 부분은 RT-task 로 돌려 속도와 제한된 시간을 유지하도록 하였다. RTLinux 와 로봇의 통신은 serial 로 하게 되는데, 이를 RT task 로 두었고, 로봇에 부착된 카메라에서 획득한 영상은 serial 로 RTLinux 의 buffer 에 쓰여지게 된다. 그 영상을 RTLinux 의 RT task module 에서 영상처리를 담당하여 모니터에 결과를 보여주게 된다.

로봇이 움직이는 동안에 카메라에 잡힌 영상의 전송과 처리는 그 제한된 시간에 반드시 처리하여야 하

는 부분이다. 이 부분에서 RTLinux 의 hard-real time 지원이 중요한 역할을 함을 알 수 있었다.

ARM 프로세서가 Floating point 연산이 불가능 하여 정밀한 영상처리는 불가능 하였으나, 영상에서 object 의 line 을 찾거나, 물체를 인식하는 정도는 충분히 가능하였다.

5. RT module 간 상호 우선순위

RT module 은 크게 2 가지로 나누었다. 첫째, 사용자의 입력을 받는 input 부분인 mouse input module 로써 Camera 로부터 획득한 영상을 잡는 부분과 Robot 의 동작을 멈추고 재개하는 역할을 담당하는 부분으로 실시간성을 보장 받아야 하기 때문이다. 둘째, ARM board 와 로봇의 시리얼 통신을 담당하는 부분이다. Camera 에서 획득한 영상을 로봇에서 ARM board 로 전송하는 동작과, 로봇의 동작을 멈추고 재개하는 명령을 ARM board 에서 로봇으로 전송하는 역할을 하는 module 이다.

그림 6 에서 이 동작을 도식화 하였다. RS232 전송 module 이 영상을 전송하는 작업 중 이더라도 mouse input 이 들어오게 되면 즉시 작업을 멈추고 마우스의 입력 명령을 먼저 수행하게 된다.

Linux 에서 동작하는 thread 는 GUI 와 영상처리를 담당하게 된다. RS232 로 전송되는 이미지는 하드웨어적으로 buffer 에 쓰이기 되므로 특별히 module 로 구현하지는 않았다. 영상을 포착하여 이미지 처리하는 부분은 Linux 의 thread 를 사용하여 구현하였다. 이들의 상호 동작은 그림 6 에서 보는 것과 같이 RT FIFO 로 통신을 할 수 있다

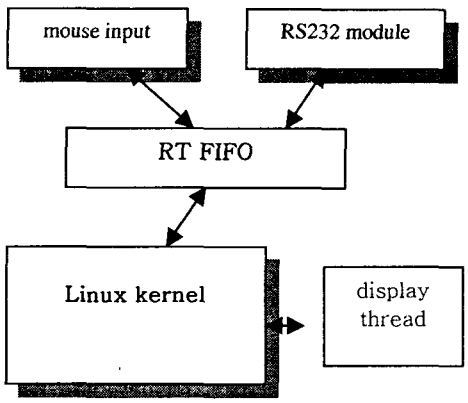

그림 6 RT module 간 상호 메커니즘

6. 결론

본 논문에서는 실시간 운영체제를 ARM board 에 올리고, 영상처리 모듈을 사용해 로봇의 카메라에서 보내오는 영상을 처리하고자 하였다. 실시간 시스템인 RTLinux 를 영상처리 모듈이 올라간 ARM board 에서 영상처리와 영상전송 부분을 Real-Time module 을 이용하여 구현하고, 제한된 시간 안에 원하는 결과가 나오도록 구현하였다.

또한 본 논문에서 사용한 시스템은 최소한의 시스템 장비를 구축하여 효과적인 실시간 제어를 구축할 수 있다는 점을 입증하였다.

더불어 로봇에 장착된 카메라에서 획득한 영상의 이미지 처리 과정을 보여 줌으로써 이 분야의 새로운 가능성을 보여 주었다.

향후 계획은 현재 serial 로 로봇과 ARM 의 통신을 담당하였는데 이 부분을 무선으로 시도하고자 한다. 현재 나온 무선의 속도와 로봇의 속도 비교를 하면 충분히 제어가 가능하리라 본다

참고문헌

[1]Michael Barabanov, " A Linux-based RealTime Operating System" Master thesis, New Mexico Institute of Mining and Technology, June 1997.

[2]Daniel P. Bovet and Marco Cesati: Understanding Linux Kernel. O' Reilly, Jan 2001

[3]이명근, 이상정, 조성범, 임재용, " 실시간처리 리눅스 기반 VoIP 시스템 설계 및 구현" , 정보과학회 2001 년 동계학술발표논문집, pp.345-351, 2001

[4]심재홍, 정석용, 강봉직, 최경희, 정기현 " Real Time Linux 시스템을 위한 재구성 가능한 메모리 할당.모델" 정보처리학회 2001 년 추계학회 논문지 A

[5]Intel Crop. SA-1110 Microprocessor developer's Manual,June 2000. Order no:278240-003

[6]이석, 문승빈," ARM CPU 를 이용한 리눅스기반 독립형 Vision 처리 모듈 개발" 정보처리학회 춘계학술 발표 논문집, pp657-660, 2002.

[7]SCOTT J.NORTON, MARK D.DIPASQUALE : The multithreaded programming guide THREADTIME. HEWLETT PACKARD, 1996

[8]Alessandro Rubini and Jonathan Corber: Linux Device Driver 2nd. O' Reilly, Jun 2001.