legOS(LEGO Operating System)의 커널 분석 및 수정을 통한 RCX간의 Routing 구현

이호익[0] 이대성 김기창
인하대학교 전자계산공학과
hihi@super.inha.ac.kr blue@super.inha.ac.kr kchang@inha.ac.kr

An implementation of routing among the RCX machine through legOS kernel analysis and modification.

Ho-Ick Lee[0] Dae-Sung Lee Ki-Chang Kim
Dept. of Computer Science and Engineering, Inha University

요 약

최근 임베디드 운영체제의 중요성이 부각되지만, 임베디드 운영체제의 개발자는 매우 열악한 상황이다. 이에 본 논문에서는 쉽게 접할 수 있는 임베디드 운영체제인 legOS를 소개하고 커널을 분석해보며, legOS의 커널 수정을 통해 RCX간의 통신 수단인 적외선 통신의 한계점을 개선하여, 적외선 통신이 불가능한 RCX간에도 통신이 가능하도록 Routing 기능을 구현하도록 한다.

1. 서 론

legOS는 MindStorms에 들어있는 RCX의 임베디드 운영체제이다[1][2]. MindStorms이란 LEGO사의 제품 중 하나로서, 여러가지 레고 부품들과 RCX라는 LEGO사와 MIT가 공동개발한 마이크로 컴퓨터를 포함하고 있다[3]. legOS라는 운영체제는 1998년 10월에 Markus Noga에 의해 시작 되었으며, 실제적으로 LEGO사와는 관련이 없는 비공식적인 운영체제로 오픈소스로 운영되고 있다.

RCX는 8Bit CPU와 모터 출력, 센서 입력의 제어 기능과 적외선을 이용한 통신기능까지 갖추고 있다. 물론 이 적외선 통신과 legOS를 적절히 활용한다면, 여러가지 적외선 장비들, 대표적으로 PDA와도 통신이 가능해진다[4]. MindStroms은 기본 펌웨어와, 그 펌웨어에 맞게 Windows 환경에서 RCX를 동작할 수 있게 하는 개발 환경이 제공되지만, 이 펌웨어를 legOS로 대체함으로써, RCX는 보다 강력하게 탈바꿈 된다. 기본 펌웨어를 RCX에 적재하면, LCD 창에 숫자밖에 표시가 안되지만, legOS를 적재하게 되면 숫자는 물론, 영문까지 표시가 가능해진다. legOS로 인하여 RCX는 보다 많은 메모리를 확보하여, 사용자 프로그램의 공간이 늘어나며, 선점형 멀티태스킹, POSIX 세마포어를 이용한 프로세스 동기화, 동적 메모리 관리, 향상된 IR 네트워킹이 가능해진다. 물론 오픈 소스인 legOS를 개선, 수정하게 된다면, 보다 많은 일이 가능해진다. legOS는 gcc 크로스 컴파일러를 이용하여 개발되므로, 기존의 그래픽한 개발 환경 대신, 텍스트 c기반의 개발환경으로 바뀌게 된다.

이에 본 논문에서는 리버스 엔지니어링(이미 나와있는 제품에 대한 분석과 연구)으로 이미 발표된 내용을 바탕으로 legOS를 분석하고[5][6][7], legOS의 수정을 통한 RCX간의 Routing 기능을 구현하여, 직접적인 통신이 불가능한 RCX간에 통신이 가능하도록 하였다.

본 논문의 구성은 2장에서 legOS가 탑재될, RCX의 Hardware에 대해 알아보고, legOS의 소스 코드를 분석하며, 3장에서는 RCX간의 Routing 기능을 위해 legOS를 설계 및 수정한 부분을 설명하였다. 4장에서는 결론 및 향후 과제에 대해 설명하고 끝을 맺는다.

2. 관련연구

본 장에서는 분석된 legOS(버전 0.2.6)를 각 부분별로 소개하고자 한다.

2.1 Hardware

RCX는 HITACHI의 H8/3292 8Bit CPU를 장착하고 있으며, H8/3292 CPU는 R0H,R0L~R7H,R7L의 16개의 8비트 레지스터가 있다[8]. 이 레지스터는 R0~R7의 8개의 16비트 레지스터로도 이용될 수 있다. R7레지스터는 스택포인터(SP) 레지스터로 이용되어, 스택의 처음을 가리킨다. 기억장치는 16K의 ROM과 외부 32K의 RAM으로 구성되어있다. ROM은 0x0000-0x7FFF 주소 공간으로 참조되며, LEGO에서 제공하는 소프트웨어를 포함하고 있다. 중요한것은 ROM의 Interrupt Handler는 RAM 내의 주소를 호출하도록 되어있다. 즉, RAM내의 특정 Interrupt Handler에 해당하는 특정 주소에 실행되기 원하는 코드를 넣는다면, 특정 Interrupt Handler에 원하는 코드를 실행할 수 있게 된다. RAM은 0x8000-0xFFFF의 주소 공간으로 참조되며, 일부 주소 공간은 LCD, Motor, Interrupt Vector, On-chip의 메모리 맵 I/O를 위해 미리 예약되어 있다. RAM은 예약된 주소 공간을 제외하고, 하위 공간을 차지하는 Kernel 영역과 나머지 공간을 차지하는 User 영역으로 구분되어진다.

그 외의 장치는 16비트 타이머, 8비트 타이머 모듈 2채널, A/D컨버터, I/O포트가 있다. 16비트 타이머는 legOS를 구동하고 관리하는데 사용되기 때문에 매우 중요하다. 나머지 2개의 8비트 타이머는 스피커의 신호와 적외선 통신의 타이밍을 만들어내고, A/D컨버터는 센서로 신호 감지를 위해, I/O포트는 버튼과 모터를 제어하기 위해 사용된다.

2.2 Kernel의 동작구조

Kernel은 가장 먼저 메모리를 초기화하며, 그 외의 필요한 장치들을 초기화 한다. 시스템 타임을 초기화 할때는 16비트 timer에 1ms 마다 timer interrupt이 발생하도록 하게하고, ROM은 timer interrupt이 발생했을 경우, ocia_vector가 가리키는 함수를 호출하도록 한다. ocia_vector는 systime_handler를 가리키게 되고, 실제적으로 1ms 마다 systime_handler가

호출된다. systime_handler는 RCX의 여러가지 장치들을 모니터링하면서 해당되는 핸들러를 호출한다. systime_handler의 마지막에서는 태스크 관리를 위한 tm_switcher를 호출하고, tm_switcher는 scheduling vector에 해당하는 함수를 호출해 프로세스를 스케쥴링 한다[9]. 그러나 RCX의 On/Off 키가 눌려지기 전까지 scheduling vector는 dummy handler를 가리키고 있어, 아무일도 일어나지 않다가, On/Off 키가 눌려지면 scheduling vector에 프로세스를 스케쥴하는 tm_scheduler를 연결해 실제적으로 프로세스 스케쥴링이 시작되 멀티태스킹이 일어난다.

2.3 Kernel의 시작

ROM은 처음에 RAM의 첫번째 시작 번지를 호출하도록 되어 있다. legOS에서는 항상 RAM의 첫번째 시작 번지인 0x8000에 kmain()함수를 위치시켜 kmain()이 호출되도록 한다. kmain()은 #ifdef 에 따라 여러가지 초기화를 진행한다. #ifdef 문은 include/config.h에 정의되어 있다. 초기화하는 동안 tm_init()에서는 커널 글로벌로서 정의되는 pd_single 프로세스 구조체를 초기화하고, 최초의 멀티태스킹용 프로세스인 tm_idle_task가 execi에 의해서 생성된다. program_init() 함수에서는 적외선 통신 패킷 처리를 위한 프로세스인 packet_consumer와 버튼 상태 체크를 위한 key_hanler가 execi에 의해 생성된다. On/Off키가 눌려질 때 까지 기다리고 있다가 systime_handler에 의해 On/Off키가 감지되면, tm_start()를 호출해 멀티 태스킹이 구현 되면서, Kernel이 작업을 시작한다.

2.4 Timing

타이밍의 초기화는 systime_init() 함수에 의해서 초기화 된다. systime_init()에서는 시스템 타이머 변수를 초기화하며, timer interrupt로 발생되는 task scheduler vector를 dummy handler로 가리키게한다. 또한, Timer Control/Status Register, Time Control Register, Time Output Control Register, Timer Output Compare Register A, Timer Interrupt Enable Register를 수정하여, 1 ms 마다 timer interrupt를 발생시킨다. 또한, timer interrupt vector인 ocia_vector를 systime_handler를 가리키도록 한다.

2.5 Task Management

legOS는 타임 슬라이스를 이용한 선점형 멀티태스킹을 제공한다. 태스크를 관리하는 방식으로는 tm_scheduler()에 의해, 먼저 우선순위가 높은 태스크들을 라운드 로빈 방식으로 실행권을 할당하고, 실행권이 주어진 태스크는 타임 슬라이스가 경과할때마다, 라운드 로빈에 따른 다음 태스크와 context switch를 하게 된다. 현재 우선순위에서 실행권이 주어질만한 적당한 태스크가 없다면, 다음 우선순위로 넘어가 실행권을 할당한다. 태스크는 자신만의 프로세스 구조체를 가지며, 프로세스 구조체에는 현재 태스크의 상태 정보와 스택 주소를 저장한다. 태스크의 상태는 Running, Sleeping, Waiting, Zombie, Dead의 5가지 상태로 구별된다. 프로세스 구조체는 execi() 함수에 의해서 생성되며, 구조체 생성시 해당 우선 순위에 자신의 태스크를 등록하며, 스택 메모리를 할당하게 된다[10].

2.6 Memory Management

legOS는 메모리를 단순하게 관리한다. 메모리의 초기화는 mm_init()로부터 시작되며, 이 함수는 특정 목적으로 사용하기 위한 메모리 부분들을 예약해놓으며, 그외에는 빈곳으로 설정한다. 메모리 블록은 4byte의 헤더와 데이터로 구성되어 진다. 메모리의 할당은 malloc()함수에 의해 구현되며, 메모리의 첫번째 비어있는 블록을 가리키는 전역 변수인 mm_first_free로부터 비어있는 블럭을 찾아 할당한다. 메모리의 해제는 free()함수에 의해 구현되며, 지정된 블록 헤더의 처음 2byte를 MM_FREE(0)로 저장함으로 해제된다.

2.7 Semaphore

세마포어의 초기화는 sem_init()에 의해 초기화 되며, sem_wait(), sem_post()에 의해서 관리된다. sem_wait()함수는 해당 세마포어값을 검사하여, 0보다 작을 경우, 바로 리턴하며, 0일 경우, 세마포어의 값이 0보다 작아질때까지 기다리게 된다. sem_post()함수는 해당 세마포어값을 1 감소 시킨다. 세마포어 값을 변경시키는 동안에 legOS는 모든 인터럽트를 중지시키고, 변경 후, 인터럽트를 재개한다.

2.8 Networking

legOS는 통신 프로토콜로 LNP(LegOS Networking Protocol)을 사용한다.

①	②	③	④	⑤	⑥
1	1	1	1	0~255 또는 0~253	1

① : 패킷의 종류. f0 - Integrity, f1 - Addressing
② : Data의 길이. Checksum 길이는 넣지 않는다.
③ : Destination Address. Addressing 패킷의 경우에만 쓰인다. Integrity 일 경우, 여기서부터 Data가 된다.
④ : Source Address
⑤ : Data
⑥ : Checksum

[그림 1] LNP 패킷의 구조

LNP는 Broadcast 성격의 Integrity 패킷과 특정 주소에만 전달하는 Addressing 패킷으로 나눌 수 있다. 패킷을 수신하게 되면, Receive Shift Register, Receive Data Register가 변경되어, Receive-end interrupt(RXI) vector인 rx_handler를 호출하게 된다. rx_handler는 collision, checksum등을 계산하고, 패킷의 종류에 따라, Integrity 또는 Addressing Handler를 호출하고, 호출된 Handler는 유저영역의 버퍼로 수신된 데이터를 복사한 후, 세마포어값의 변경으로 유저에게 수신된 패킷이 있음을 알린다. 패킷의 송신은 버퍼로 송신할 패킷을 복사한 후, Serial Control Register를 변경하여, Send interrupt vector인 tx_handler를 호출하여, 버퍼의 내용을 전송하게 된다. legOS는 자신의 패킷이 아닐 경우에는, 버리도록 되어있으므로, 다른 RCX로부터의 패킷 중계(Routing)가 불가능하다.

3. RCX간의 Routing 기능 구현

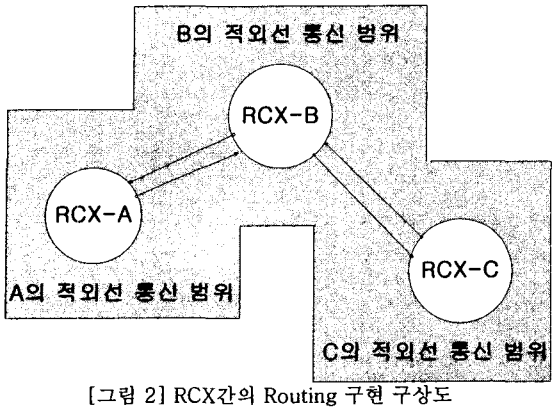
RCX는 네트워킹을 위해 적외선 통신을 사용하고 있다. 그러나 적외선 통신은 거리와 방향에 한계가 있기 때문에, RCX간의 통신시 적외선의 방향이나 거리에서 벗어나면 통신이 불가능해진다. 따라서, 본 논문에서는 통신을 원하는 RCX끼리 서로 통신이 불가능한 곳에 있더라도, 중계 역할을 해주는 RCX만 있다면, 통신이 가능하리라는 생각에서 legOS를 수정하여, legOS에 중계(Routing)기능을 구현하여 보았다.

legOS에서 Routing 기능을 구현하기 위해서는 legOS가 사용하고 있는 통신 프로토콜인 LNP를 수정하고, 수정된 LNP 패킷에 따라 legOS가 처리할 수 있도록 해야한다.

3.1 구현된 Routing 기능 처리과정

A가 C의 데이터를 원할 경우, A는 C와 직접적으로 통신이 불가능하므로, B가 A와 C의 중계 역할을 해주어야 한다.

1. A가 자신의 주위에서 Routing 가능한 RCX를 찾는다.
2. B는 A의 Routing요청을 받았으므로, Routing 준비가 되었다는 응답을 보낸다.
3. A는 Routing 가능한 B를 찾았으므로, C의 데이터를 요청한다.



[그림 2] RCX간의 Routing 구현 구상도

3. B는 A의 요청 데이터가 C의 것임을 확인 후, 주위에서 Routing 가능한 RCX를 찾는다.
4. C는 B의 Routing요청을 받았으므로, Routing 준비가 되었다는 응답을 보낸다.
5. B는 Routing 가능한 C를 찾았으므로, A가 보낸 C의 데이터를 요청한다.
6. C는 B가 요청한 데이터가 자신의 것임을 확인 후, 데이터를 B에게 전달한다.
7. B는 C에게 전달 받은 데이터를 A에게 전달한다.

✱ 각 RCX에 바퀴를 달아 회전이 가능하도록 하였다.

3.2 Routing을 위한 LNP프로토콜의 수정

LNP프로토콜은 [3.1 Routing 기능 처리과정]을 수행할 수 있도록 수정되었다. 패킷의 종류에 'f2'를 추가함으로서, Routing 패킷임을 구별하고, Routing Header에 따라서 routing의 종류를 달리 하였다.

①	②	③	④	⑤	⑥	⑦	⑧	⑨
1	1	1	1	1	1	1	0~250	1

① : 패킷의 종류. f0 - Integrity, f1 - Addressing, f2 - Routing
② : Routing Data의 길이. Checksum 길이는 넣지 않는다.
③ : Routing Header
ping, pingAck, reqData, repData, noData
④ : Destination Address
⑤ : Source Address
⑥ : *Not used*
⑦ : 최초의 요청 Source Address
⑧ : Data
⑨ : Checksum

[그림 3] Routing 기능이 추가된 LNP 패킷의 구조

3.3 Routing 기능을 위한 Kernel 수정

Routing 기능을 구현하기 위해서는 legOS가 패킷 수신시 Rouing 패킷을 처리하도록 하여야 하고, 각 Routing 패킷에 따라 이에 상응하는 패킷을 만들어 응답해주어야 한다. 또한 Routing 정보를 저장할 자료구조가 필요하다.

```
struct route_peer_info_t {
    char routeyName[20];        /* Machine's name */
    char routeMyData[20];       /* My routing data */
    unsigned char dest;         /* Destination address */
    unsigned char src;          /* My address */
    unsigned char o_src;        /* Original source address */
    char data[6];               /* Request routing data */
    int len;                    /* Data length */
};
```

[자료구조 1] Routing 자료 구조

Routing 패킷 처리를 위해, 패킷 수신을 처리하는 lnp_receive_packet()함수에 Routing packet(f2)처리를 하였다.

1. 최초의 패킷 수신 처리 함수인 lnp_receive_packet()에서 routing 패킷을 분류한다.
2. 분류된 routing 패킷에 따라서 패킷의 정보를 routing 자료 구조에 저장하고,요청에 대한 적절한 응답 패킷을 만들어 보낸다.
   1) ping 요청일 경우, pingAck 응답.
   2) reqData 요청일 경우, 자신의 데이터 요청이면 repData 후 종료, 아니라면 주위의 RCX에게 ping 요청.
   3) pingAck가 오면, reqData 요청
   4) repData가 오면 1)에서 요청 온 RCX에게 repData전달. noData 가 오면, noData 전달. 후 종료
3. 종료 후, routing 자료구조 초기화 및 패킷 정보 초기화한다.

[알고리즘 1] Routing 패킷 수신 처리 과정

1. 수신된 패킷의 정보가 있는 routing 자료 구조를 참조하여 패킷 종류, routing header를 포함한 임시 버퍼를 생성한다.
2. 임시 버퍼의 내용을 Destination RCX로 보낸다.
3. 임시버퍼를 삭제한다.

[알고리즘 2] Routing 패킷 송신 처리 과정

```
//응답이 오거나, timeout 될때 까지
while(replyPingData) {
      motor_a_dir(rev);  // 왼쪽 바퀴 뒤로 회전 시작
    motor_c_dir(fwd); // 오른쪽 바퀴 앞으로 회전 시작.
    // RCX가 회전함
};
```

[알고리즘 3] Routing 가능한 RCX 찾기 과정

3.4 Routing 기능 구현 결과

Routing 기능이 구현된 3대의 RCX를 [그림2]처럼 배치한 후, A에게 C의 데이터를 요청하도록 하였다. A의 요청에 따라 B가 회전하면서, C의 위치를 확인한 후, C로 부터 데이터를 전송 받아 A에게 성공적으로 전해주었다.

4. 결론 및 향후과제

본 논문에서 legOS의 커널을 분석하고, 커널의 미진한 부분을 보완하고자 Routing 기능을 추가하였다. Routing 기능이 legOS에 추가됨으로서, 직접 접근이 불가능한 RCX와도 통신이 가능하게 되었지만, 현재 여러 RCX와의 병렬적으로 Routing 기능은 되어 있지 않고, 응답이 없을 경우에 대한 처리가 미흡하므로, 이에 대한 구현이 있어야 하겠다.

참고문헌

[1] LEGO. " ROBOTICS INVENTION SYSTEM" http://mindstorms.lego.com/, 2002
[2] Markus Noga. " About legOS" http://www.noga.de/legOS/, 1998
[3] MIT Media-LAB. " Mindstrorms and RCX Q&A" http://fredm.www.media.mit.edu/people/fredm/mindstorms/
[4] Kyosuke, " Playing with LEGO MINDSTORMS" . http://www.asahi-net.or.jp/~qx5k-iskw/lego/index.html, 2002
[5] Kekoa proudfoot, Stanford CA-LAB. " RCX Internals" . http://graphics.stanford.edu/~kekoa/rcx/, 1998, 1999
[6] Stig Nielson." Introduction to the legOS Kernel" , 2000
[7] legOS News group. " legOS News" http://www.lugnet.com/robotics/rcx/legos/, 2002
[8] HITACHI. " H8/3292 CPU HARDWARE MANUAL 3rd Edition"
[9] KIP R.IRVINE. " 어셈블리 언어-인텔계열-3판" , Prentice Hall · 교보문고
[10] Diniel P.Bovet, Marco Cesati." Understanding the Linux Kernel" , O' REILLY · 한빛미디어