Journal of the Korea Industrial Information Systems Research Vol. 20 No.6, Dec. 2015 :21-27
http://dx.doi.org./10.9723/jksiis.2015.20.6.021 ISSN:1229-3741

리눅스 9비트 시리얼통신에서 모드전환 지연원인의 분석과 개선

(Diagnosis and Improvement of mode transition delay in Linux 9bit serial communications)

정 승 호[1], 김 상 민[2], 안 희 준[3]*
(Seungho Jeong, Sangmin Kim, and Heejune Ahn)

요 약 본 논문은 리눅스 환경에서 9비트 RS-232 통신에 필요한 패러티 모드 전환 방식을 사용할 때 발생하는 바이트 간 전송 지연증가 문제를 분석하고 해결책을 제시한다. 문자 전송방식인 RS-232통신에서 메시지의 시작을 나타내기 위하여 9비트통신을 하는 경우가 상당히 있다. 8 비트 문자통신을 기본으로 하는 통상의 리눅스에서는 9비트지원을 하기위해서는 패러티 모드를 변환하는 방법이 사용되는데, 실험결과 이때 OS 틱(tick) 수준의 지연이 발생하는 것을 확인하였다. 본 논문에서 지연의 원인이 드라이버에서 전송 FIFO 버퍼에 남은 데이터를 기다리는데 걸리는 시간의 최소단위를 OS 틱을 사용하기 때문인 것을 밝혀내었으며, 표준 리눅스 드라이버를 수정하여 패러티 모드전환 시간을 1ms 이내로 감소시켰다. 최근 다양한 시스템 통신 방식의 개발되었지만, 여전히 기존의 많은 표준 및 시스템이 RS-232 방식을 사용하여 9 bit 통신을 하고 잇는 경우에 리눅스 활용이 가능하게 되었다는 의미가 있다.

핵심주제어 : 리눅스, 시리얼 드라이버, 패러티 모드, 9 bit 통신, 통신지연, 프로토콜

Abstract We analyze the problem that is occurring when using parity mode transformation required for 9 bit serial communication under Linux environment and propose the solution. The parity mode change is used for 9 bit serial communication in the Linux that by nature supports only 8 bit serial communication. delay (around OS tick) arises. Our analysis shows that the cause is minimum length of waiting time to transmit data remained in Tx FIFO buffers. A modified Linux serial driver proposed in this paper decreases the delay less than 1ms by using accurate time delaying. Despite various system communication interfaces, enormous existing standards and system have adopted RS-232 serial communication, and the part of them have communicated by 9bit serial.

Key Words : Linux, 9 bit serial communication, parity mode, inter-byte delay, protocol specification, real-time

* Corresponding Author : heejune@snut.ac.kr
† 이 연구는 서울과학기술대학교 교내연구비의 지원으로 수행되었습니다(2015-1447).
Manuscript receivrd December 12, 2015 / revised December 22, 2015 / accepted December 24, 2015
1) (주)LS산전, 제1저자
2) (주)몽태랑, 제2저자
3) 서울과학기술대학교 전기정보공학과, 교신저자

1. 서 론

RS-232[1] 시리얼 통신방식은 1960년대 개발된 이후로 최근까지 산업계에서 장치 간 대표적인 통신방식으로 사용되어왔다. RS-232방식이 이렇게 장기간 넓은 활용된 이유는 통신방식의 단순함 때문이다. 송수신 라인만으로도 연결이 가능하고, 비동기방식에 스펙이 까다롭지 않아 소형장치에서도 쉽게 구성이 가능하다[2]. 하지만 상대적으로 저속이며, 일대일 연결이라는 점 등의 단점을 가지고 있다. 이 때문에 근래에 들어서는 시스템 내부 통신은 I2C나 SPI 등의 사용도 증가하고 있고, 시스템 간의 통신은 Ethernet이나 CAN 버스 등을 사용하는 방식이 증가하고 있다. 그러나 기존의 많은 표준들과 제품들이 RS-232를 사용하고 있기 때문에 여전히 산업적인 활용도가 높은 통신 방식이다.

RS-232C 시리얼통신은 8-bit 데이터를 사용하는 경우가 가장 보편적인 방법이나, 9-bit 데이터를 사용하는 경우가 있다[3]. 주요 사례는 문자전송방식인 RS-232 방식을 통하여 프레임 전송을 할 때, 프레임을 구분하기위하여 첫 번째 바이트를 표시는 경우이다. 비트에 9번째 비트를, 수신노드들을 깨우기 위한 비트라는 의미에서 'wake-up' 비트라고 부른다. 프레임을 구분하는 일반적인 방법으로는 프레임 시작 시 STX바이트를 추가하는 방법[4]이 있으나, 이 경우 시작문자와 데이터를 구별하기위한 회피(ESCAPE)처리로 인하여 동작이 복잡해지고, 단일 바이트로 메시지 전송인 경우 지연이 생기는 단점이 있다. 9 bit 통신을 지원하지위해서는 현재의 컴퓨터 시스템은 바이트단위에 맞춰져서 설계가 되었기 때문에 예외적인 기술이 사용되어야한다. 이러한 문제는 학계에서 연구로는 자주 다루어지고 있지 않으나, 인터넷 검색을 해보면 (예, 구글에서 "serial 9 bits"), 산업계에서는 이에 대한 수요와 중요성이 큰 것을 확인 할 수 있다. 본 연구에서 모든 응용사례를 들기는 어려우므로, 본 저자들이 슬롯머신을 관리하는 국제 표준 프로토콜인 SAS (slot accounting system) [5]을 구현하면서 접한 경우를 대상으로 문제점과 해결책을 제시하였다.

기본적인 해결방법은 제2절에 자세히 설명하겠지만, 패러티 비트를 데이터비트로 차용하여 사용하는 방식이다[6]. 이 방식을 적용하여 리눅스 환경에서 실험한 결과 기능적인 면에서는 9bit 통신을 만족하는 것을 확인하였다. 그러나 일반적인 리눅스 환경에서 패러티모드 변환 시에 바이트간격시간이 과대하게 길어지는 것이 관측되었다. 이러한 문제를 확인하기위하여 제3절에서는 리눅스 드라이버 분석과 실험을 바탕으로 그 원인을 분석한다. 이를 바탕으로 제4절에서 바이트 시간간격 문제를 해결하기위한 디바이스 드라이버의 수정과 실험결과를 보인다. 제 5절에 논문의 의미에 대하여 살펴본다.

2. 패러티 비트를 이용한 시리얼 9bit 데이터 통신 방법

다음 그림 1은 8bit 데이터를 사용하는 UART 통신 방식의 예를 보여준다. 이때 parity를 사용하는 방식에 따라서, no-parity, mark, space, even-parity, odd-parity로 나눈다. (표 1)

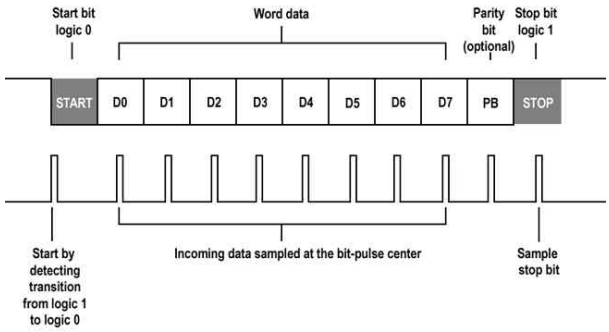

Fig. 1 Serial communications protocol example

Table 1 UART Parity mode

Problems	Percentage (%)
Numerical control error	40.5
Machining center error	35.5
Pick up error	20.0
Stacker crane error	4.0
Total	100.0

패러티 비트를 데이트비트로 차용하여 사용하는 방법을 사용하면, 리눅스 및 윈도우즈와 같이 데이터비트의 길이가 8비트를 지원하는 환경에서 9비트 통신이 가능하다. 즉, 9번째 비트가 0이면 패러티에 0을 1이면 패러티 비트에도 0이 세팅되도록 모드 변환을 하는 방식을 사용한다. 따라서 직접적인 방법은 mark 패러티와 space 패러티를 사용하는 것이다. PC 하드웨어 환경에서는 이러한 방식을 사용하면 된다. 하지만, 대부분의 임베디드 리눅스환경에서는 mark, space 패러티가 드라이버에서 구현이 안 되어 있는 경우가 많다. 이는 상대적으로 사용 빈도가 떨어지기 때문이라고 생각된다. 이런 경우는 그림 2와 같이 전송할 데이터의 1의 비트수와 even, odd parity 조합하여 사용하는 방식을 통하여 해결이 가능하다. 예를 들어 b000010101을 전송하려면, 8 비트에 해당하는 b00001010을 데이터로 나머지 b1을 패러티비트로 전송하면 되는데, b00001010이 짝수개의 1을 가지고 있으므로, odd 패러티 모드를 사용하면 하드웨어가 마지막 비트를 1로 자동으로 세팅하게 된다. 반면 b000010100을 전송하기위해서는 even 패러티를 세팅하면 된다. 여러 개의 데이터를 연속적으로 전송할 때는, 이전 패러티 모드와 향후 패러티 모드가 다른 경우에만 모드 전환을 하면 된다.

리눅스 환경에서 이를 구현할 때는 터미널 장치의 모드을 조절하기위해서는 리눅스 터미널 프로그래밍 인터페이스인 'tcsetattr'를 사용하여야 한다[7]. 그림 2은 패러티 모드를 mark와 space로 세팅하는 C 코드 예를 보여 준다. 이때 특히 주목할 부분은 "TCSADRAIN" 옵션의 사용하는 부분이다. 시스템은 전송지연이 있기 때문에 'write' 호출이후에도 전송이 안 된 데이터가 있을 수 있다. 남은 데이터가 있을 때 패러티 모드 전환을 요청하게 되면, 다음 세 가지 옵션중 하나를 선택하여야한다. 바로 모드를 전환하고 1) 남은 데이터를 전환된 모드로 전송 (TCSANOW) 한다. 2) 남은 데이터를 버퍼에서 지우고 모드 변환 (TCSAFLUSH) 한다. 3) 기존의 데이터를 다 전송할 때 까지 기다린다. (TCSADRAIN) 본 연구에서는 모든 데이터가 원하는 설정 모드로 전송하기위해서 'TCSADRAIN'를 사용하여야한다.

```
void setParityMode(
    int fd,
    bool mode /*true: odd, false: even*/ )
{
    struct termios tio;
    bzero(&tio, sizeof(tio));
    tcgetattr(fd, &tio);
    if(mode == false)
        tio.c_cflag &= ~PARODD;
    else
        tio.c_cflag |= PARODD;
    tcsetattr(fd, TCSADRAIN, &tio);
    /*
    notice "TCSADRAIN" should be used
    */
}
bool oddevenTable[256] = { false, true,  . . .
} /* true if the number of 1's is odd */
void sendData(
    int fd,
    unsigned data,
    bool paritybit)
{
    static bool currentMode = false;
    bool  requiredMode =
        oddevenTable[data] ^ paritybit;
    if(currentMode == requiredMode ) {
        requiredMode);
    }
    write(fd, data, 1));
}
```

Fig. 2 Parity mode control code in Linux environment

3. 실험결과

그림 4는 2절에서 설계한 방식으로, 라즈베리-파이2를 시스템으로 사용한 환경에서 SAS 프로토콜의 대표적인 메시지를 송수신하는 경우를 보여준다. 9-bit 데이터 통신하는 이 가능함을 보인다. 그러나 동작의 시작적인 특성을 살펴보면, 모드 전환하는 경우에 바이트 간 시간 간격이 크

게 증가되어 있음을 볼 수 있다. 저자들이 확인한 바로는 정도의 차이는 있으나 일반 PC, 산업용 PC, 그리고 업계에서 많이 사용되는 freescale사나, broadcom사의 임베디드 리눅스 환경에서 모두 동일한 문제가 있음을 확인하였다.

대부분의 프로토콜은 전송 바이트간격에 시간 제약이 있다. 예를 들어 SAS 프로토콜의 경우는 5ms를 최대 바이트 간격으로 하고 있다. 즉, 프레임 내부에 바이트들 사이의 간격이 5ms이상이 되면 프레임은 에러로 처리된다. 따라서 모드 전환의 시간은 5ms보다는 작아야한다.

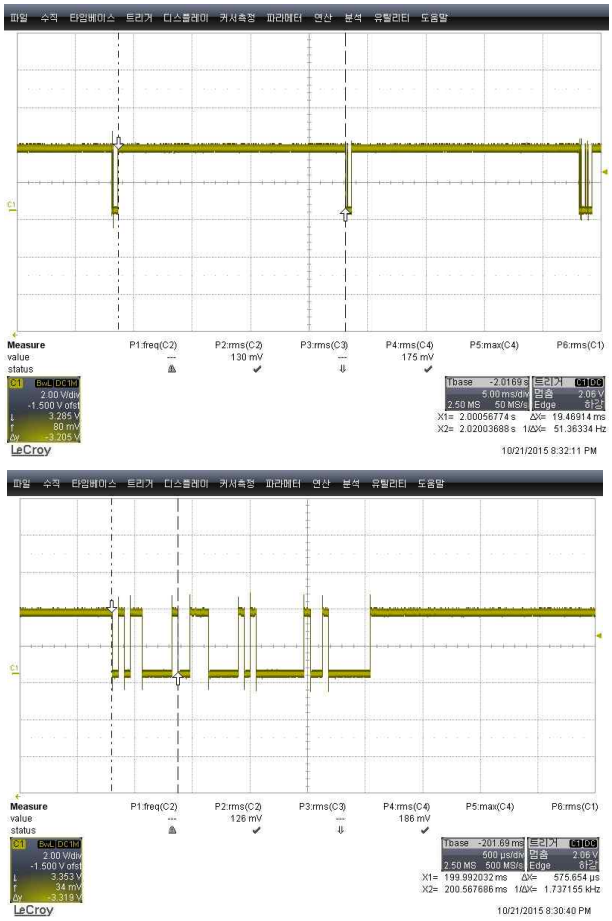

Fig. 3 mode change figures

그림 4의 검증을 위하여 사용한 실험환경을 보여준다. 데이터의 송수신과 시간을 확인하기위하여 상용 RS-232 테스트 장치[8]를 사용하여 확인하였다. 전송속도는 19200 baudrate로 하고 데이터는 1부터 14 까지를 반복하여 전송하였고, 모두 space 패러티를 사용한다고 가정하였다. 이때 앞선 데이터와 전송할 데이터의 바이너리 표현에서 1의 수가 다르면 모드 전환을 해야 한다. 그림 5의 상위 그래프는 데이터의 전송을 시간적으로 그래프로 표시한 것이다. 그림에서 지연 시간이 현저히 늘어난 부분은 parity 모드가 변경되는 부분 (그래프에 적색으로 표시됨)이다. 이는 리눅스를 기반으로 하는 여러 시스템에서 출력으로 확인하여도 동일한 결과가 확인된다.

이러한 문제의 원인은 시스템 부하가 원인은 아닌 것으로 확인할 수 있다. 저자들은 다른 프로그램이 실행되지 않는 상황에서 CPU부하가 10%이하인 상황에서도 동일한 결과를 확인하였다. 리눅스 기본 코드를 확인한 결과 기본적으로 사용되는 시리얼 드라이버에 모드 전환 시에 기존의 전송 버퍼의 데이터가 전송되기를 기다리는 함수인 uart_wait_until_sent()의 구현에서 실제 기다리는 시간을 계산할 때 최소 1 OS tick (jiffies) 만큼 지연되도록 설계되어 있기 때문으로 확인 되었다. msleep_interruptible() 커널 함수 자체는 수 밀리초를 지원하고 있으나, 계산방식에서 양자화가 적용되어 문제가 생긴 것으로 보인다[9]. OS tick은 일반적인 임베디드 시스템의 경우 10ms를 사용하고 있다. 또한 OS tick을 1m로 줄인 경우에도 바이트 지연이 5ms 이하로 줄어들지 않는 것을 확인하였다.

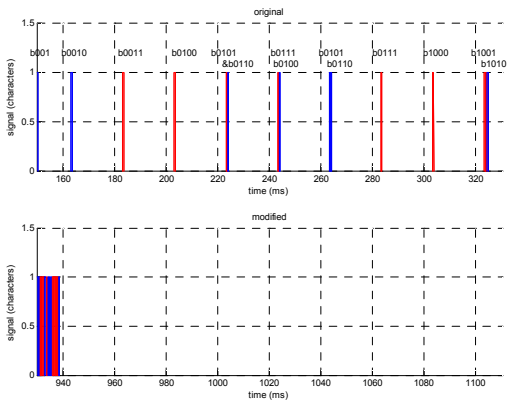

Fig. 4 Timing example by Serial monitoring tool (top: original, bottom: modified)

```
static void uart_wait_until_sent(
    struct tty_struct *tty, int timeout)
{
// omitted

    char_time =
        (port->timeout - HZ/50) /
            port->fifosize;
    char_time = char_time / 5;
    if (char_time == 0)
        char_time = 1;
    if (timeout && timeout < char_time)
        char_time = timeout;

// omitted

    while (!port->ops->tx_empty(port)) {
    msleep_interruptible(
        jiffies_to_msecs(char_time));
        //waiting for empty
    if (signal_pending(current))
        break;
    if (time_after(jiffies, expire))
        break;
    }
}
```

Fig. 5 Linux serial driver
(/drivers/tty/serial/serial_core.c version 4.3)

4. 시간 지연의 해결 방법

저자들이 지적한 문제 상황과 그 원인을 바탕으로 해결책은 두 가지가 가능하다. 하나는 새로운 메시지 기반 드라이버를 사용하는 방식이로(현재 미국 바운더리 디바이스 등 몇 몇 업체에서 사용하고 있는 방법), 이고 나머지 하나는 드라이버에 정교한 시간계산을 하도록 수정하는 방식이다. 전자의 방식은 사용자 프로그램에서 드라이버로 메시지 단위로 데이터를 전달하고, 드라이버가 모드 변환을 진행하는 방식인데, 일반적인 Linux 터미널 API 사용방식과는 다르므로 프로그램을 수정하여야한다. 따라서, 본 논문에서는 리눅스 API 표준을 그대로 유지하는 후자의 방식을 사용하였다. 그 결과 수정된 드라이버의 부분은 다음과 같다.

```
static void uart_wait_until_sent(
    struct tty_struct *tty, int timeout)
{
// omitted

    char_time =
        (port->timeout - HZ/50) /
            port->fifosize;
    char_time = char_time / 5;
    if (char_time == 0)
        char_time = 1;  //sleep_flag
    if (timeout && timeout < char_time)
        char_time = timeout;

// omitted

    while (!port->ops->tx_empty(port)) {
        if (uart_chars_in_buffer(tty) <= 1)
      // sleep only if 1B in TX buffer >1.
            break;
        msleep_interruptible(
            jiffies_to_msecs(char_time));
            //waiting for empty
        if (signal_pending(current))
            break;
        if (time_after(jiffies, expire))
            break;
    }
}
```

Fig. 6 Modified Linux serial driver

그림 6의 수정된 드라이버의 아이디어는 다음과 같다. 시리얼 장치의 한 바이트 전송시간은 OS 틱에 비하여 상당히 작다. 예를 들어 19200 baudrate인 경우 약 0.5m. 따라서 전송이 시작한 경우 지연시간은 일반적으로 기다려야하는 시간은 매우 작으므로, msleep_inteeruptible을 생략하는 것이 바람직하다. 이는 이 경우 바로 패러티

모드전환을 요청하더라도 시리얼 장치제어 구조상 패러티 모드의 전환은 다음 전송 바이트부터 적용되기 때문이다. 그림 4의 아래 그래프는 새롭게 구현된 드라이버를 바탕으로 기존과 동일한 환경에서 실험한 결과를 보여준다. 그래프 상에서 보는 바와 같이 모드전환이 필요한 경우에도 지연이 1 ms이하로 줄어든 것을 확인할 수 있다. 이로인하여여 전체적으로 데이터 전송 속도가 향상되는 효과도 보였다. 표 2은 10분간 리눅스 시리얼드라이버를 수정하기 전과 수정한 후의 지연시간 차이를 통계적으로 정리한 것이다. 전체 지연 시간의 평균이 감소하여 제안한 방식으로 수정하는 것이 메시지 전송에 긍정적인 효과가 있음을 알 수 있다. 그리고 패러티 모드가 변하지 않을 때는 지연시간이 수정전과 수정 후에 차이가 없는 것을 통해 패러티모드가 변하지 않는 반면, 패러티모드의 전환이 있을때 는 평균 18.69 ms의 큰 지연감소 효과가 있음을 확인했다.

Table. 2 Comparison of average inter-byte times between original and modified Linux drivers

inter-byte time	Original Linux Driver	Modified Linux Driver	Difference
no mode change	0.573 ms	0.573 ms	same
mode change	19.299 ms	0.609 ms	18.690 ms reduced
average	11.942 ms	0.595 ms	11.347 ms reduced

또한, 드라이버 변환으로 인한 태스크 스케줄링 상의 문제가 발생하는지를 보기위한 로드테스트 실험으로, 웹브라우저 등을 실행하여 시스템에 부하를 준 상태에서 동일한 실험을 반복한 결과, 모드전환의 지연시간에 전혀 차이가 없음을 확인 하였다.

5. 결 론

RS-232 통신은 오랫동안 장치 간 통신방식으로 애용되어 왔으며, 앞으로도 오랜 기간 사용될 것으로 보인다. 최근에는 라즈베리-파이와 같은 2-3만원대 이하의 리눅스 기반 시스템이 산업계에서 많이 활용되고 있다 [11, 12, 13]. 본 연구에서는 리눅스 환경에서 9비트 통신을 구현하기위한 방안에 대하여 분석하고 해결책을 제시하였다. 리눅스환경에서 바이트 단위의 패러티 모드 전환을 사용하는 경우 1ms이하의 실시간 성이 요구된다. 리눅스가 그 동안 실시간처리용으로 기능을 개선하여 왔지만, 범용 환경에서는 수 ms의 지연을 보장하는 실시간 처리에는 어려움이 있다. 따라서 9-bit 통신방식에 시간적인 제약이 발생한다. 본 연구에서 표준 시리얼 드라이버를 다른 응용에 문제가 없도록 최소한으로 수정하여 9 bit 통신에 있어 문제를 해결하였다. 그 결과 저자들이 사용하는 응용 프로토콜을 포함한 유사한 9bit 통신이 가능하게 되었다. 이러한 결과는 실용적인 측면에서 의미가 있다고 생각 되면, 수정 코드를 GNU 리눅스 조직에 기고하여 코드에 반영을 요청할 예정이다.

본 연구에서는 Ubuntu나 라즈베리와 같은 일반적인 리눅스 시스템 환경만을 고려하였고, 실시간 Linux 등에 대하여는 충분한 비교를 하고 있지 못하다. 또한 몇몇 상용 임베디드 리눅스 업체에서 제공하는 9비트용 패킷단위 전송 드라이버를 작성하는 방법과도 자세한 비교 연구가 필요하다.

References

[1] EIA standard RS-232-C: Interface between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange. Washington: Electronic Industries Association. Engineering Dept. 1969.

[2] RS-232, Wikipedia, https://en.wikipedia.org/wiki/RS-232

[3] SealLevel, "What Is 9-Bit Data Framing?", Aug. 2010 (available at http://www.sealevel.com/support/article/AA-00146/0/What-Is-9-Bit-Data-Framing.html)

[4] James F. Kurose, Keith W. Ross, "Computer Networking: A Top-Down Approach (6th Edition)." *(book)*
[5] IGT, Slot Accounting System, version 6.01 June, 2003
[6] H. Ahn, and SM Kim, "Design and Implementation of Casino Slot machine Accounting Protocol Analysis System," Journal of Korean Society for Computer Game, Vol. 26, No.2 pp. 35-41, 2013. *(journal)*
[7] Kerrisk, Michael. The Linux programming interface. No Starch Press, 2010. *(book)*
[8] RS232 Sniffer, EX-TAP http://www.stratusengineering.com/EZTap.html (retrived 2010. 10. 10)
[9] CrossRef, Linux Kernel source, http://lxr.free-electrons.com/source/drivers/tty/serial/serial_core.c
[10] Love, Robert. Linux kernel development. Pearson Education, 2010. *(book)*
[11] S.-Y. Heo, W.-J. Lee, B.-J. Shin, K.-J. Han, "Design and implementation of packet fitlering mechanism for secure TeredoService)", Journal of the Korea Industrial Information System Society, Vol. 12, No. 3, pp. 47-59, 2007. *(journal)*
[12] D. Y. Kim, J. B. Kim, S. Y. Rhew, "Performnace verification process for introduction of Open Source Software - centered on introduction of Linux into the NEIS," Journal of the Korea Industrial Information System Society, Vol. 11, No. 3, pp. 59-68, 2006. *(journal)*
[13] J. Lee, S. Oh, K. Chung, T Yun, K Ahn, "I/O performance analysis about memory allocation of the UBIFS," Journal of the Korea Industrial Information System Society, Vol. 18, No. 4, pp. 9-18, 2013/ *(journal)*

정 승 호 (Seungho Jeong)

- 비회원
- 서울과학기술대학교 컴퓨터공학과 공학학사
- 서울과학기술대학교 제어계측공학과 공학석사
- 관심분야 : 임베디드 소프트웨어, 컴퓨터 통신, 멀티미디어

김 상 민 (Sangmin Kim)

- 정회원
- 서울과학기술대학교 전기정보공학과 공학사
- 서울과학기술대학교 전기정보공학과 공학석사
- (주) 몽태랑 기술연구소 선임연구원
- 관심분야 : 카지노전산시스템, 임베디드 소프트웨어

안 희 준 (Heejune Ahn)

- 정회원
- KAIST 전기및전자공학과 공학박사
- 서울과학기술대학교 정보통신대학 전기정보공학과 정교수
- 관심분야: 인터넷 컴퓨팅, 임베디드 소프트웨어