임베디드 리눅스에서 프로세스 우선순위를 고려한 실시간 통신 지원

진현욱° 이상헌 윤연지
건국대학교 정보통신대학 컴퓨터공학부
°jinh@konkuk.ac.kr

Real-Time Communication Support based on Process Priority for Embedded Linux

Hyun-Wook Jin° Sang-Hun Lee Yeon-Ji Yun
Department of Computer Science and Engineering, Konkuk University

요 약

프로세스의 우선순위는 임베디드 시스템에서 수행되는 여러 가지 작업들의 중요도 또는 작업 마감 시간의 임박성에 따라 결정되기 때문에 프로세스의 입출력 작업 시에도 반영되어야 한다. 하지만 많은 임베디드 운영체제들은 기존의 범용 운영체제를 기반으로 설계되었기 때문에 입출력 작업 시 프로세스의 운선순위를 반영하지 못하고 있다. 본 논문에서는 이러한 문제를 해결하기 위해서 새로운 통신 프로토콜 스택 구조를 제안하고 이를 임베디드 리눅스에 구현한다. 또한 본 논문은 이더넷이 산업용 기기 등의 연결에 활용될 수 있음에 주목하고 독립 이더넷 네트워크에 적합한 전송 프로토콜을 제안한다. 측정 결과 제안된 프로토콜 스택 RTDiP(Real-Time Direct Protocol)은 UDP/IP와 비교하여 단방향 통신 지연시간을 최대 59% 감소시켰으며 통신 처리율을 최대 155% 향상시킬 수 있음을 보인다. 또한 낮은 우선순위를 갖는 배경 통신 프로세스에 의해서 UDP/IP는 532%나 단방향 통신 지연시간이 증가하나, RTDiP은 2% 미만의 증가만을 보임으로써 프로세스의 우선순위에 따라 패킷 처리가 이루어지고 이를 통해서 실시간 통신을 지원해줄 수 있음을 보인다.

1. 서 론

임베디드 운영체제들은 실시간성 지원을 위해서 여러 가지 프로세스 스케줄링 기법들을 사용한다 [1]. 그 예로서 널리 사용되고 있는 임베디드 리눅스의 경우에는 프로세스의 우선순위(priority)를 기반으로 FIFO(First-In-First-Out), RR(Round Robin)과 같은 실시간 스케줄링을 지원한다 [2]. 프로세스의 우선순위는 임베디드 시스템에서 수행되는 여러 가지 작업들의 중요도 또는 작업 마감 시간(deadline)의 임박성에 따라 결정되기 때문에 프로세스의 입출력 작업 시에도 반영되어야 한다.

입출력 작업은 일반적으로 예측이 힘든 작업 시간을 요구하며 그 요청이 특정 시점에 집중적으로 발생하는 특성을 갖기 때문에 실시간성을 지원해 주기 위해서는 특히 조심스럽게 설계되어야 한다. 하지만 많은 임베디드 운영체제들은 기존의 범용 운영체제를 기반으로 설계되었기 때문에 입출력 작업 시 프로세스의 운선순위를 반영하지 못하고 있다. 특히 대부분의 임베디드 시스템이 네트워크에 연결되어 있는 것을 고려한다면, 네트워크 입출력에 있어서 프로세스의 우선순위 반영은 전체 시스템의 실시간성 보장을 위해서 중요한 요소라고 할 수 있다. 네트워크 시스템에서 프로세스의 우선순위 반영이 힘든 가장 근본적인 원인중의 하나는 bottom half를 기반으로 한 운영체제의 설계에 있다. bottom half는 사용자 프로세스의 우선순위보다 높기 때문에 실질적인 통신 프로토콜 처리는 프로세스의 우선순위와는 무관하게 이루어진다.

본 논문에서는 이러한 문제를 해결하기 위해서 새로운 통신 프로토콜 스택 구조를 제안하고 이를 임베디드 리눅스에 구현한다. 제안된 프로토콜 스택 구조는 프로세스의 우선순위에 따른 통신 프로토콜 처리가 가능하도록 한다. 그 결과 우선순위가 높은 프로세스의 실시간성을 보장해줄 수 있음을 보인다.

또한 본 논문은 이더넷(Ethernet)[3]이 산업용 기기 등의 연결에 활용될 수 있음에 주목한다. 이더넷은 다른 산업용 네트워크에 비해서 고속 고대역폭을 제공할 수 있는 장점을 갖고 있다. 산업용 및 무인 기기들은 많은 경우 독립 네트워크를 구성한다 [4][5][6]. 따라서 이더넷의 상위에서 사용되는 인터넷 대상의 기존 TCP/IP 프

이 논문은 2007년도 한국전자통신연구원 위탁과제 (#7010-2007-0008) 지원에 의한 논문임.

로토콜 스택은 부적합할 수 있다. 본 논문은 이러한 문제점을 해결하기 위해서 제안된 프로토콜 스택 구조를 기반으로 독립 이더넷 네트워크에 적합한 전송 프로토콜을 제안한다. 제안된 전송 프로토콜은 UDP/IP에 비해서 고속 고대역폭을 제공하고 각각의 패킷에 우선순위를 명시할 수 있다.

본 논문은 다음과 같이 구성되어 있다. 본 서론에 이어 2장에서는 기존 프로토콜 스택의 구조가 프로세스의 우선순위를 반영하기에 부족한 이유에 대해서 설명하고, 관련 연구들에 대해서 토의한다. 3장에서는 프로세스의 우선순위를 반영할 수 있는 새로운 통신 프로토콜 스택 구조를 제안한다. 4장에서는 제안된 프로토콜 스택을 기존의 UDP/IP와 비교 측정한다. 마지막으로 5장에서 본 논문의 결론 및 향후계획을 기술한다.

2. 연구 배경

본 장에서는 우선 기존 임베디드 리눅스가 프로세스의 우선순위를 네트워크 패킷 처리에 반영하지 못하는 구조적인 문제점을 지적한다. 그리고 관련연구에 대해서 토의한다.

2.1 프로세스의 우선순위와 통신

본 절에서는 임베디드 리눅스 커널 버전 2.6을 기반으로 이더넷을 통한 기존 프로토콜 스택의 네트워크 패킷 처리에 대해서 설명한다.

우선 송신 프로세스가 시스템 호출(system call)을 사용하여 데이터 전송을 요청하면, 사용자 버퍼에 있는 데이터는 커널 버퍼로 복사되어지고, 전송 계층은 체크섬 계산을 수행한다. 데이터 전송을 위한 패킷 헤더 작성 후, 패킷은 디바이스 드라이버에 의해서 네트워크 컨트롤러에게 전달된다. 패킷이 전달된 후 송신 작업이 종료되어 시스템 호출이 반환된다.

수신의 경우는 조금 더 복잡한 과정을 거친다. 패킷 수신을 알리는 인터럽트가 발생되면 인터럽트 핸들러가 패킷을 커널 버퍼로의 수신을 완료하고 bottom half에게 처리할 데이터가 있음을 알린다. bottom half는 수신측에서 네트워크 계층과 전송 계층의 패킷 처리를 수행한다. 이후에 수신 프로세스는 해당 데이터를 사용자 버퍼에 복사함으로써 수신을 완료한다.

이와 같이 기존의 프로토콜 스택은 패킷 처리 시에 해당 패킷을 송수신하는 프로세스의 우선순위는 전혀 고려하지 않는다. 즉 모든 패킷들을 단순히 송수신된 순서만을 기반으로 처리하고 해당 프로세스에게 전달할 뿐이다. 또한 수신 시 실행되는 bottom half는 네트워크 계층과 전송 계층의 대부분 작업을 수행하는 커널 수준의 함수로서 인터럽트 핸들러보다 낮은 우선순위를 갖지만 사용자 프로세스들보다는 높은 우선수위를 갖는다. 따라서 낮은 우선순위를 갖는 프로세스가 수신할 패킷들을 bottom half가 처리할 경우에는 높은 우선순위를 갖는 프로세스는 수행되지 않는다.

이와 같이 임베디드 리눅스는 기존 범용 운영체제의 네트워크 프로토콜 스택 구조를 그대로 가져옴으로써 실시간성 지원에 한계를 갖고 있다. 네트워크 프로토콜 스택이 커널에 구현되어 있다고 하더라도 단순히 모든 프로세스보다 높은 우선순위를 갖고 패킷들을 처리하기 보다는 각 패킷을 수신할 프로세스의 우선순위를 고려하여 처리한다면 프로세스들의 실시간성 요구를 더욱 만족시킬 수 있을 것이다.

2.2 관련 연구

사용자 프로세스들의 실시간 통신 요구를 만족시키기 위해서 많은 연구들이 진행되었다. 별도의 통신 프로세스를 두고 이들이 커널 수준이 아닌 사용자 수준에서 패킷 처리를 수행하는 방안이 제안되었으나 [7][8], 해당 송수신 프로세스의 우선순위에 맞게 지속적으로 이들 통신 프로세스의 우선순위를 변화시켜야 하는 문제점이 있다. 또한 통신 프로세스와 응용 프로세스간의 데이터 이동은 큰 지연시간을 초래한다.

또 다른 해결책으로 upcall 기반의 실시간 통신 지원이 제안되었다 [9]. 이 제안은 패킷의 수신을 알리기 위해서 커널에 구현된 upcall을 사용하며, 등록된 upcall 핸들러의 우선순위에 따라 upcall의 발생 순서를 결정한다. 하지만 upcall 핸들러의 우선순위는 프로세스의 우선순위와 별도로 관리되는 문제점을 갖고 있다. upcall 핸들러의 우선순위를 초기에 프로세스의 우선순위와 일치시킨다고 해도 프로세스 스케줄러가 프로세스의 우선순위를 동적으로 변화시킨다면 이를 반영해야 하는 문제점을 갖고 있다.

사용자 수준의 미들웨어에서 프로세스 스케줄링을 하는 기법 역시 제안되었다 [10]. 하지만 이러한 가법은 커널 내부에 구현되어 있는 스케줄러 외에 별도의 프로세스 스케줄러를 사용자 수준에 구현해야 한다. 따라서 커널 내부에 구현되어 있는 프로토콜 스택에서 이들 프로세스의 우선순위를 반영하지 못한다면 지적된 한계를 극복할 수 없다.

이더넷으로 독립 네트워크를 구성하고 실시간 통신을 제공하기 위한 여러 가지 노력과 상용화가 이루어지고 있다 [11][12][13]. 하지만 이러한 실시간 이더넷 통신에서 프로세스의 우선순위를 고려한 패킷 처리에 대한 고찰은 아직 이루어지지 않고 있다.

즉 현재까지 실시간 통신을 위해서 많은 연구들이 진행되었지만, 수행 도중에 동적으로 변경될 수 있는 프로세스의 우선순위를 반영한 효율적인 실시간 통신 지원에 대한 연구는 이루어지지 않았다. 이러한 면에서 본 논문의 연구는 기존의 연구와 차별화된다고 할 수 있다.

3. 프로세스의 우선순위를 고려한 실시간 통신

본 장에서는 프로세스의 우선순위를 반영하여, 높은 우선순위를 갖는 프로세스에 속하는 네트워크 패킷을 먼저 처리할 수 있는 프로토콜 스택 구조를 제안한다. 또한 제안된 프로토콜 스택은 독립 이더넷 네트워크에 적합하도록 설계한다. 제안된 프로토콜 스택은

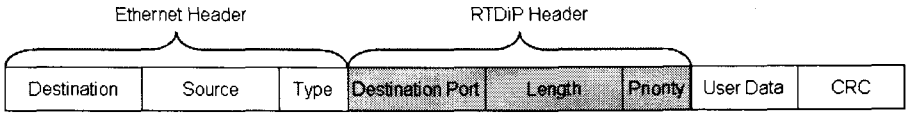

그림 1 RTDiP 패킷 헤더 구조

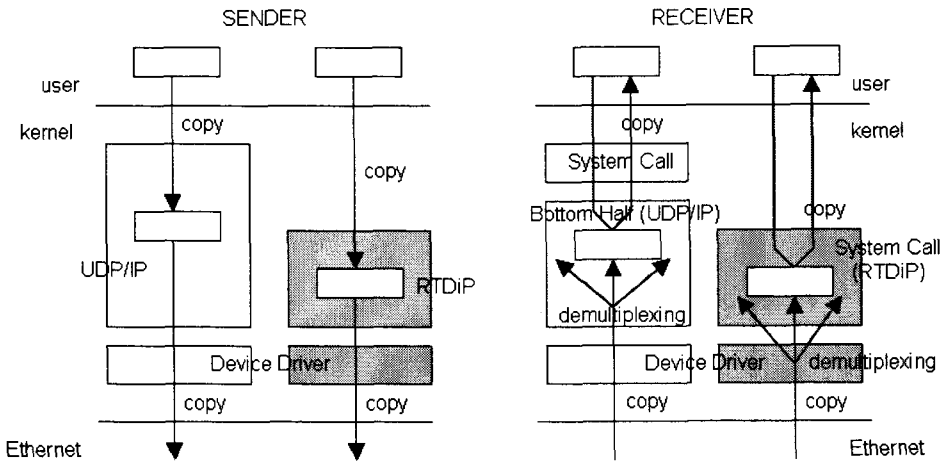

그림 2 RTDiP과 UDP/IP의 프로토콜 스택 구조 비교

RTDiP(Real-Time Direct Protocol)으로 불린다.

본 논문에서는 산업용 기기의 연결을 위한 독립 네트워크를 구성하는 이더넷을 그 대상으로 하고 있다. 반면 기존에 널리 사용되고 있는 TCP/IP는 인터넷 환경을 대상으로 하기 때문에 많은 오버헤드를 포함하고 있다. 예를 들어서 송수신 노드를 명시하기 위해서 단일 네트워크 환경에서는 이더넷의 MAC 주소만으로 충분하다. 따라서 IP주소는 생략할 수 있다.

본 논문에서 제안되는 RTDiP은 이러한 TCP/IP의 오버헤드를 피하기 위해서 이더넷 계층 바로 위에 전송 계층을 구현한다. 전송 계층은 각각의 연결(connection)을 구분하기 위한 포토 번호를 갖고 있다. 또한 각 패킷들의 우선순위를 명시할 수 있도록 하였다. 이들 정보를 저장하기 위한 헤더의 구조는 그림 1에 보이고 있다.

포트번호(그림 1의 Destination Port)는 각각의 프로세스가 RTDiP 연결을 설정할 때 명시되며, 하나의 프로세스는 여러 개의 포트번호를 사용할 수 있다. 수신측에서 기존의 프로토콜 스택은 bottom half가 전송 계층의 헤더를 보고 해당 패킷의 수신 프로세스를 결정한다. 이와 다르게 RTDiP에서는 디바이스 드라이버가 포트번호를 보고 최종 수신 프로세스를 결정한다. 디바이스 드라이버에서의 추가적인 지연시간을 최소화하기 위해서 다른 패킷 처리에 해당하는 작업은 수행하지 않는다.

이후 RTDiP 패킷의 실질적인 처리 및 사용자 버퍼로의 복사는 사용자 프로세스의 시스템 호출에 의해서 수행되며, 별도의 bottom half를 구동하지 않는다. 사용자 프로세스는 각자의 우선순위에 따라서 실행 기회를 갖기 때문에 우선순위가 높은 프로세스는 수신완료를 위한 시스템 호출을 수행할 기회를 우선순위가 낮은 프로세스에 비해서 더 많이 갖게 된다. 따라서 별도의 스케줄러 구현 또는 수정 없이 기존의 프로세스 스케줄러에 의해서 프로세스의 우선순위에 따라 수신 패킷의 처리가 이루어진다. 또한 스케줄러의 정책에 따라서 프로세스의 우선순위가 동적으로 변화한다고 해도 자연스럽게 높은 우선순위를 갖게 된 프로세스의 해당 패킷을 먼저 처리하게 된다. 이와 같이 패킷들은 기본적으로 프로세스의 우선순위에 의해서 그 처리 순서가 결정되나 같은 프로세스에 수신되는 패킷들 간에는 헤더에 명시되어 있는 패킷 우선순위(그림 1의 Priority)가 높은 것부터 처리된다. 현재의 RTDiP 구현은 세 단계(high, medium, low)의 패킷 우선순위를 제공한다.

그림 2는 설명된 RTDiP의 전체 구조와 기존의 프로토콜 스택(UDP/IP)의 구조를 비교하고 있다. 그림에서 볼 수 있는 바와 같이 RTDiP은 이더넷 기반의 단독 네트워크를 대상으로 하기 때문에 오버헤드가 큰 기존의 전송(UDP) 계층과 네트워크(IP) 계층을 생략하고 바로 이더넷을 통해서 통신이 이루어짐을 알 수 있다. 또한 수신측에서는 디바이스 드라이버가 역다중화(demultiplexing)를 수행하고 있으며, bottom half 없이 시스템 호출로 수신을 완료함을 보여주고 있다.

제안된 RTDiP은 커널의 수정 없이 디바이스 드라이버의 수정만으로 구현이 가능하다. 또한 프로세스 스케줄링 정책에 의존적이지 않으므로 기존의 모든 실시간 스케줄러 위에서 프로세스의 우선순위를 반영할 수 있는 구조를 제공한다.

4. 성능 측정

본 장에서는 3장에서 제안된 RTDiP과 UDP/IP의 성능을 측정하고 비교한다. 기본적인 통신 성능을 비교하기 위해서는 단방향 지연시간과 통신 처리율을 측정한다. 그리고 프로세스의 우선순위에 따른 실시간성 보장을 보여주기 위한 성능을 측정한다.

RTDiP은 리눅스 커널 버전 2.6.9에 포함되어 있는 LAN91C111 이더넷 컨트롤러의 디바이스 드라이버(smc91x.c)를 수정하여 구현했다. 성능 측정을 위해서 Intel Xscale PXA270 (520MHz)을 장착하고 있는 휴인스의 Acumen270 프로세서 보드[14]를 사용하였다. 네트워크 성능 측정을 위해서 두 개의 보드는 이더넷 스위치 또는 허브 없이 이더넷 케이블로 바로 연결하였다.

4.1 단방향 지연시간 및 통신 처리율 측정

단방향 지연시간을 위해서 우선 왕복 지연시간을 측정한다. 특정 크기의 데이터를 송신하고 수신측이 이를 다시 송신자에게 보냄으로써 데이터의 왕복 통신 지연시간을 측정할 수 있다. 측정된 왕복 지연시간을 반으로 나눔으로써 단방향 지연시간을 구한다.

그림 3은 단방향 지연시간의 측정 결과를 보여주고 있다. 그림에서 볼 수 있는 바와 같이 모든 데이터 크기에 대해서 RTDiP이 UDP/IP보다 낮은 지연시간을 가짐을 알 수 있다. 작은 크기의 데이터에 대해서 RTDiP이 낮은 지연시간을 보이는 것은 3장에서 언급되었듯이 독립 네트워크를 위한 필수 기능만을 구현함으로써 프로토콜 처리 오버헤드를 크게 줄였기 때문이다. 큰 데이터에 대해서 낮은 지연시간을 보이는 주요 원인은 RTDiP은 추가적인 체크섬 계산을 수행하지 않음으로써 바이트당 오버헤드(per-byte overhead)[15]를 감소시켰기 때문이다. 이와 같이 독립 네트워크를 위해서 설계된 RTDiP은 UDP/IP의 단방향 지연시간을 최대 59% 감소시켰다.

통신 처리율 측정은 다음과 수행했다. 송신 프로세스는 주어진 윈도우 크기만큼의 패킷을 송신하고 이들을 수신 완료한 수신 프로세스는 수신 확인 메시지(ack)를 송신측으로 발송한다. 송신측은 ack 수신 후에 윈도우 크기만큼의 패킷을 다시 발송한다. 주어진 회수만큼 위의 작업을 반복한 후에 수신된 총 데이터 크기를 전체 통신 시간으로 나누어 Kbps의 성능을 기록한다.

그림 4는 통신 처리율 측정 결과를 보여주고 있다. 통신 처리율 역시 RTDiP이 UDP/IP보다 우수한 성능을 보이고 있다. 이는 앞에서 언급했듯이 RTDiP이 독립 네트워크에서 효율적인 통신이 가능하도록 설계되었기 때문이다. 그 결과 UDP/IP에 비해서 최대 155% 높은 통신 처리율을 보이고 있다. 기록된 최대 통신 처리율(5234Kbps)은 측정 도구의 윈도우 기반 흐름 제어를 향상시키면 더욱 높아질 수 있을 것이다.

4.2 실시간성 측정

4.1절에서 보였듯이 RTDiP은 이더넷 기반의 독립 네트워크에서 고속 고대역폭을 지원하도록 설계되었다. 본 절에서는 프로세스 우선순위에 따라 패킷 처리의 우선순위를 결정하는 RTDiP의 실시간 통신 지원 기능을 측정한다. 이러한 특성을 측정하기 위해서 4.1절에서 사용한 지연시간 측정 도구와 처리율 측정 도구를 동시에 실행시킨다. 이때 지연시간 측정 프로세스의 우선수위는 높게 설정하고, 통신 처리율 측정 프로세스의 우선순위는 낮게 설정한다. 따라서 프로세스의 우선순위에 따라 네트워크 패킷 처리가 수행된다면 측정된 지연시간은 4.1절에서 기록된 지연시간과 큰 차이가 없어야 한다.

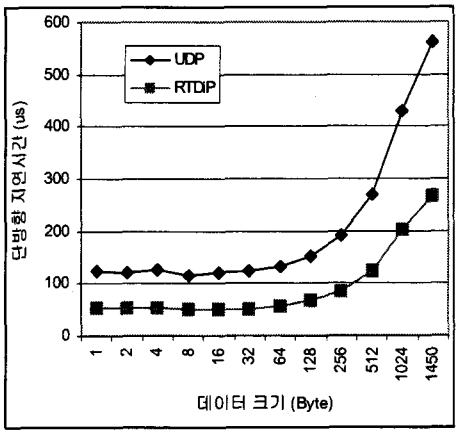

그림 3 단방향 지연시간 비교

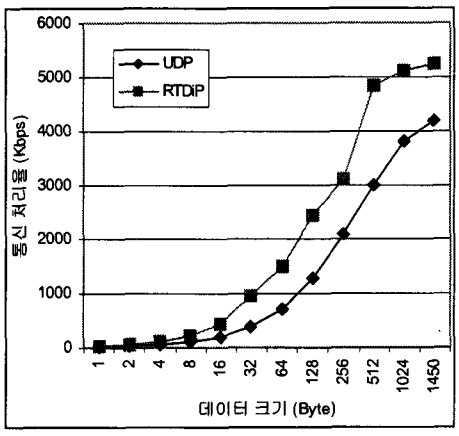

그림 4 통신 처리율 비교

그림 5는 RTDiP의 단방향 지연시간 측정 결과를 보여주고 있다. 그림에서 "without background"는 4.1절에서의 측정 결과로서 단방향 지연시간 측정 프로세스 외의 배경 통신 프로세스가 없는 경우다. 그리고 그림에서 "with background"로 표시되어 있는 값은 낮은 우선순위를 갖는 통신 처리율 측정 프로세스를 함께 실행시킨 경우에 기록된 단방향 지연시간을 나타낸다. 그림에서 알 수 있듯이 RTDiP의 단방향 지연시간은 우선순위가 낮은 배경 통신 프로세스에 의해서 2% 미만의 증가만을 보이

고 있다. 따라서 그림에서는 두 경우의 값 차이가 거의 확인되지 않고 있다. 이것은 RTDiP이 네트워크 패킷을 송수신할 프로세스의 우선순위에 따라 처리할 수 있음을 보여주는 것이다.

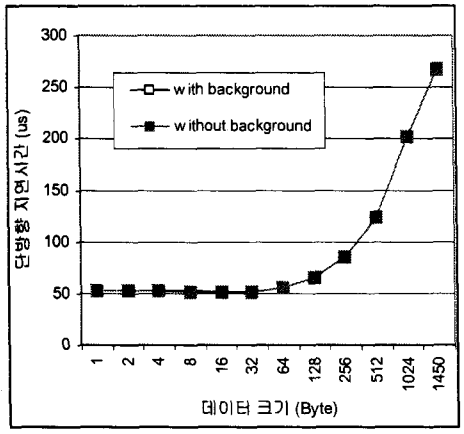

그림 5 RTDiP의 실시간성 측정

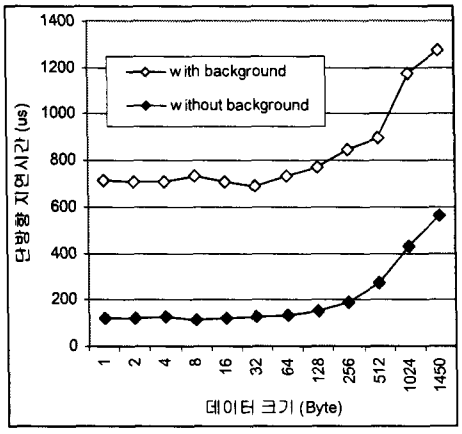

그림 6 UDP/IP의 실시간성 측정

그림 6은 UDP/IP의 단방향 지연시간 측정 결과를 보여주고 있다. UDP/IP는 RTDiP과는 다르게 배경 통신 프로세스의 유무에 따라서 지연시간이 크게 변화하는 것을 볼 수 있다. 실험에서 생성된 배경 프로세스는 대용량의 데이터를 전송하지만 그 우선순위는 낮다. 2.1절에서 언급되었듯이 UDP/IP 프로토콜 스택은 프로세스의 우선순위와는 무관하게 송수신 순서대로 패킷을 처리한다. 또한 수신측의 bottom half는 모든 사용자 프로세스 보다 높은 우선순위를 갖기 때문에 bottom half가 낮은 우선순위를 갖는 프로세스에게 전달될 패킷을 처리하면 우선순위가 높은 프로세스의 통신이 지연된다. 그 결과 그림 6과 같이 배경 통신 프로세스의 우선순위가 낮음에도 불구하고 높은 우선순위를 갖는 지연시간 측정 프로세스에게 큰 영향을 미치고 지연시간을 최대 532%나 증가시키고 있다.

5. 결론 및 향후계획

본 논문은 기존 임베디드 리눅스가 프로세스의 우선순위에 대한 고려 없이 네트워크 패킷을 처리하는 문제점을 해결하고 이더넷 기반의 독립 네트워크에서 효율적인 통신을 지원하기 위한 프로토콜 스택 RTDiP을 제안하였다. RTDiP은 송수신 프로세스의 우선순위에 따라 해당 패킷들이 처리될 수 있도록 하기 위해서 디바이스 드라이버가 역다중화를 수행하고 bottom half 없이 시스템 호출에 의해서 수신을 완료하도록 하였다. 또한 독립 네트워크의 특성을 고려하여 낮은 오버헤드를 갖는 전송계층을 이더넷 계층 바로 위에 정의하였다.

측정 결과 RTDiP은 UDP/IP와 비교해서 단방향 통신 지연시간을 최대 59% 감소시켰으며, 통신 처리율을 최대 155% 향상시킬 수 있음을 보였다. 또한 낮은 우선순위를 갖는 배경 통신 프로세스에 의해서 UDP/IP는 532%나 단방향 통신 지연시간이 증가하나, RTDiP은 2% 미만의 증가만을 보임으로써 프로세스의 우선순위에 따라 패킷 처리가 이루어지고 이를 통해서 실시간 통신을 지원해줄 수 있음을 보였다.

향후 사용자 버퍼와 커널 버퍼간의 복사를 제거하여 단방향 지연시간과 통신 처리율을 더욱 향상시킬 계획이다. 또한 소켓 인터페이스 (Sockets interface)를 RTDiP의 상위에 구현하여 기존 응용 프로그램들의 수정을 피할 수 있도록 하려고 한다.

참고 문헌

[1] C. L. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of the ACM, 20(1), pp. 46-61, 1973.

[2] IEEE Std. 1003.1b-1993, IEEE Standard for Information Technology - Portable Operating System Interfaces (POSIX®) - Part 1: System Application Program Interface (API) - Amendment 1: Realtime Extension [C language], 1993.

[3] IEEE Std. 802.3: LAN/MAN CSMA/CD Access Method, December 2005.

[4] K. H. Kim, E. Henrich, C. Im, M. -C. Kim, S. -J. Kim, Y. Li, S. Liu, S. -M. Yoo, L. -C. Zheng, and Q. Zhou, "Distributed Computing Based Streaming and Play of Music Ensemble Realized Through TMO Programming," Proc. of IEEE Int'l Workshop on Object-Oriented Real-Time Dependable Systems (WORDS2005), pp. 129-138, February 2005.

[5] 박정화, 이보은, 김정국, "TMO-eCos 기반의 실시간 이족로봇 제어 프레임워크에 관한 연구", 한국정보과학회 2007 한국컴퓨터종합학술대회 논문집, 제34권, 제1호(B), 2007년 6월.

[6] 박한솔, 신찬희, 김문회, "실시간 객체 모델을 이용

한 내장형 무인항공기 제어시스템의 설계 및 구현", 한국정보과학회 2006 가을 학술발표논문집, 제33권, 제2호(A), 2006년 10월.

[7] T. Nakajima and H. Tokuda, "User-level Real-Time Network System on Microkernel-basedOperating Systems," Real-Time Systems, 14(1), pp. 45-60, 1998.

[8] K. M. Zuberi and K. G. Shin, "An Efficient End-Host Protocol Processing Architecture for Real-Time Audio and Video Traffic," Proc. Network and Operating System Support for Digital Audio and Video (NOSSDAV), July 1998.

[9] R. Gopalakrishnan and G. M. Parulkar, "A real-time upcall facility for protocol processing with QoS guarantees," Proc. of ACM Symposium on Operating Systems Principles (SOSP), 1995.

[10] C. Shen, O. González, K. Ramamritham, and I. Mizunuma, "User Level Scheduling of Communicating Real-Time Tasks," Proc. of IEEE Real Time Technology and Applications Symposium, 1999.

[11] A. Moldovansky, "Utilization of Modern Switching Technology in EtherNet/IP Networks," Proc. of International Workshop on Real-Time LANs in the Internet Age (RTLIA'2002) in conjunction with the 14th Euromicro Conference on Real-Time Systems, June 2002.

[12] EtherCAT Technology Group, "EtherCAT - Ethernet for Control Automation Technology," http://www.ethercat.org/.

[13] IEEE Std. 1588 -2002, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems."

[14] (주)휴인스, http://www.huins.com/.

[15] H. -W. Jin and C. Yoo, Impact of Protocol Overheads on Network Throughput over High-Speed Interconnects: Measurement, Analysis, and Improvement, The Journal of Supercomputing, Vol. 41, No. 1, pp. 17-40, July 2007.