論文2002-39CI-6-4

실시간 웹서버 시스템을 위한 통합 스케줄링 방안

(An Integrated Scheduling Approach for Real-Time Web Servers)

姜 奉 直 *, 丁 錫 龍 *, 李 賢 淑 **, 崔 景 熙 **, 鄭 己 鉉 ***, 柳 海 永 ****

(Bong-Jik Kang, Suk-Yong Jung, Hyun-Suk Lee, Kyung-Hee Choi, Gi-Hyun Jung, and Hae-Young Yoo)

요 약

본 논문에서는 웹서버를 탑재한 내장형 시스템에서 실시간 속성을 만족시킬 수 있는 스케줄링 기법을 제안한다. 내장형 시스템에 웹서버를 응용프로그램의 형태로 탑재하는 경우 운영체제와 웹서버간의 이원화된 스케줄링 때문에 우선 순위 역전 현상이 발생하여 실시간 속성을 만족시키지 못한다. 이에 본 논문에서는 웹서버의 일부 스케줄링 기능을 운영체제의 스케줄러와 통합 내장시켜 일관된 스케줄링 방법을 제공한다. 제안된 스케줄링 기법을 실험용 내장형 시스템에 적용한 결과, 웹서버를 일반 태스크로 구현한 경우와 달리 우선 순위 역전 현상이 발생하지 않았다. 또한 우선 순위에 따라 태스크별 응답 시간이 시스템 부하에 관계없이 일정하여 제안된 스케줄링 기법이 실시간 내장형 웹서버 시스템에 적합하다.

Abstract

This paper proposes an integrated scheduling mechanism for embedded system with real-time web server to meet the characteristics of real time task. The proposed scheduling mechanism may solve the so-called priority inversion problem in scheduling between urgent web requests and tasks with low priorities. The priority inversion problem happens because of operating two independent schedulers, web scheduler and operating system scheduler in a system without considering the requirements of each other. In the proposed mechanism, two schedulers are integrated in an operating system and the integrated scheduler schedules tasks for urgent web requests with real time characteristics and other application tasks together. Since all tasks are scheduled by one unified scheduler that knows the characteristics of tasks, the tasks are scheduled with their absolute priorities and thus the priority inversion problem can be eliminated. The performance is measured on a prototype embedded system with the proposed algorithm.

Keywords : real-time web server, embedded system, integrated scheduling

* 正會員, 東洋工業專門大學 電算經營技術工學部
(Dongyang Technical College, School of Computer Engineering and Management)
** 正會員, 亞洲大學校 情報通信專門大學院
(Ajou University, Graduate School of Information and Communication Technology)
*** 正會員, 亞洲大學校 電子工學部
(Ajou University, School of Electronic Engineering)
**** 正會員, 檀國大學校 情報컴퓨터學部
(Dankook University, Major of Computer Science/ Division of Information & Computer Science)
接受日字:2002年5月24日, 수정완료일:2002年11月8日

1. 서 론

인터넷 기술은 복잡한 통신 프로토콜과 클라이언트 프로그램을 별도로 개발할 필요가 없고 접속 장소에 제한이 없는 등 장점을 가지고 있기 때문에 다양한 응용 분야에서 표준 사용자 인터페이스 기술로 자리를 잡아가고 있다. 최근에는 웹 카메라나 실시간 디지털 비디오와 같은 내장형 시스템에 웹 기술을 적용한 인터넷 디바이스(Internet Device)들도 제안되면서 웹 브라우저를 통해 디바이스에 대한 제반 정보에 액세스함은 물론 디바이스의 제어 및 감시 기능까지도 수행한다. 이와 같이 내장형 시스템의 제어 및 감시가 웹 브라우저를 통하여 가능하려면 내장형 시스템의 제어를 위한 프로그램은 물론 웹서버와 다양한 홈페이지까지도 내장하고 있어야 하며 홈페이지와 제어 프로그램과의 연동이 필요하다.

내장형 시스템에 웹서버를 탑재하는 연구는 일반적인 내장형 운영체제 상에 경량화한 웹서버를 효율적으로 내장시키는 방향으로 진행되어 왔다[1,2]. [1]에서는 제한된 자원을 가지는 네트워크 장치에 경량 웹서버를 내장하여 관리자가 웹 브라우저를 통하여 네트워크 장치에 직접 관리 기능을 수행시킬 수 있도록 하였다. [2]에서는 원격제어용 내장형 시스템을 위한 웹서버 구조를 제안하고 있다. 제안된 구조에서는 내장형 시스템에서 발생하는 모든 이벤트를 해당 태스크로 전달하여 수행되도록 하는 이벤트 채널 메커니즘을 제안하고 있다. 또한 웹서버 자체를 운영체제 커널의 일부로 포함하려는 연구들도 진행되고 있다. *khttpd*[3]와 *phttpd*[4]는 리눅스 커널에 통합된 웹서버 모듈이다. 이 모듈들은 웹 브라우저로부터 요청되는 웹 요청을 분석한 결과가 정적인 웹 문서 요청인 경우, 즉 파일 시스템에 저장된 웹 문서를 단순히 읽어서 송신하는 경우에 커널에서 직접 파일 시스템에 접근하여 서비스한다. 그러나 웹 요청 분석 결과가 동적인 웹 문서 요청인 경우, 즉 CGI (Common Gateway Interface)와 같이 프로그램 실행 결과에 의하여 동적으로 생성되는 웹 문서를 요청하는 경우에는 커널 내부의 웹서버 모듈이 커널 외부의 웹서버에게 해당 요청을 전달한다. 웹 요청을 전달받은 커널 외부의 웹서버는 요청을 분석하고 해당되는 CGI 프로그램을 실행하여 웹 문서를 동적으로 생성하여 송신한다. *Tux*[5]도 또한 리눅스 커널에 통합된 웹서버 모듈이다. Tux는 위에 열거한 웹서버 모듈들과 달리 동적인 웹 문서 요청의 경우에도 프로그램을 직접 실행할 수 있는 커널 인터페이스를 제공한다. 이와 같은 연구들은 웹서버의 일부 기능을 운영체제 커널 내에 통합함으로써 정적인 웹 문서를 처리하는 경우에 발생하는 웹서버와 커널간의 불필요한 메모리 복사를 줄여 웹서버 성능을 향상시키는 결과를 보여준다.

이와 같이 범용 운영체제 혹은 내장형 운영체제에 웹서버를 내장하여 실행하는 방법은 웹서버를 쉽게 이식할 수 있는 등 여러 면에서 장점을 가지고 있다. 이러한 구조를 가진 시스템에서는 웹 브라우저가 보내는 명령을 담은 패킷을 운영체제가 분석하지 않는다. 그리고 웹서버에 전달되어야 비로소 패킷을 분석하고 해당 작업을 스케줄하게 된다. 따라서 웹 브라우저를 통하여 전달된 인터넷 디바이스의 제어 명령의 경우 등 실시간성을 가진 서비스가 요청된 경우에 운영체제는 패킷에 실시간 서비스를 요청하는 내용이 기록되었는지를 모르므로 요청된 서비스의 실시간성에 맞추어 웹서버를 스케줄하기가 사실상 어려워진다. 이에 따라 장치의 실시간 조작의 실패 혹은 실시간 작업의 실행 지연 등으로 인한 문제점이 발생할 수 있다.

이러한 문제점을 해결하기 위하여 본 논문에서는 단순히 웹서버를 경량화하여 내장시키는 것이 아니라 웹 태스크와 장치의 조작에 관한 내부 태스크간의 우선순위를 통합 관리할 수 있는 기법을 제안한다. 본 연구에서 인터넷 디바이스가 실시간성 서비스를 제공하는 방안은 다음과 같다. 우선 모든 인터넷 디바이스의 조작에 관한 명령 및 그 명령을 처리할 태스크를 웹 요청과 일대일 대응을 이루도록 하였다. 그리고 웹 요청을 분석하는 기능을 웹서버로부터 분리한 후 운영체제에서 이루어지도록 하고 운영체제상에서 실행되는 웹서버는 운영체제가 분석한 요청을 처리하도록 단순화하였다. 따라서 일반적인 홈페이지를 요청하는 패킷이 도착하는 경우에는 운영체제는 이를 웹서버에서 전달한다. 그러나 실시간 서비스에 관련된 홈페이지를 요청하는 패킷이 도착하는 경우에 운영체제는 패킷 분석 작업을 통하여 이를 파악하고 실시간 서비스를 처리할 태스크를 스케줄한다. 따라서 운영체제는 실시간성을 가지는 웹 요청이 도착하는 경우에도 웹서버의 도움 없이 직접 패킷을 분석하여 스케줄하기 때문에 장치의 실시간 조작 실패 혹은 실시간 작업의 실행 지연 등의 문제점을 해결할 수 있다.

웹서버로의 요청을 우선 순위에 따라 처리하는 컨넥션 스케줄링에 대한 연구[6,7]가 이루어지고 있으나 이는 일반적인 웹서버에서 서비스 품질(QoS)을 보장하기 위해 운영체제와 독립적으로 클라이언트 요청에 대한 서비스 품질을 보장하도록 하는 것이다. 또한, *Tux*, *khttpd*, 혹은 *phttpd* 등 웹서버 기능을 운영체제 커널의 일부로 포함하는 연구도 있으나 이는 단지 웹서버의 성능을 향상시킬 목적으로 커널이 직접 웹 브라우저의 요청을 분석하고 처리하는 방법이다. 따라서 본 논문에서 제안하는 스케줄의 통합과는 다르다.

본 논문의 구성은 다음과 같다. 2장에서는 일반적인 웹서버에 의하여 인터넷 디바이스가 실시간성 서비스를 처리하는 경우에 발생하는 우선 순위 역전 현상을 기술하고, 3장에서는 통합 스케줄 기법을 소개한다. 4장에서는 실시간 운영체제인 μC/OS에 패킷 분석 모듈을 추가하여 구현한 통합 스케줄러를 기술하고, 5장에서 성능 평가를 기술한 후, 6장에서 결론 및 향후 연구를 기술한다.

Ⅱ. 통합 스케줄러의 필요성

웹서버를 하나의 태스크로 구현하여 내장시킨 인터넷 디바이스들은 태스크 스케줄링이 이원화되어 있다고 볼 수 있다. 웹 브라우저를 통해서 도착되는 웹 요청들은 운영체제 커널에 의해 그대로 웹서버에게 전달되고, 웹서버가 웹 요청들을 분석하여 웹 태스크들을 스케줄한다. 그리고 디바이스 내부의 태스크들은 운영체제 커널에 의해 우선 순위에 따라 스케줄된다. 즉, 웹 태스크들은 웹서버가 스케줄하고 내부 태스크들은 커널이 스케줄하게 된다. 그런데 이와 같이 이원화된 스케줄링 환경에서는 높은 우선 순위의 웹 태스크가 낮은 우선 순위의 태스크보다 실행이 지연되는 우선 순위 역전 현상이 발생할 수 있고, 결과적으로 인터넷 디바이스의 실시간 스케줄링을 보장하지 못하는 문제점을 야기할 수 있다.

이원화된 스케줄링 환경의 인터넷 디바이스에서 발생할 수 있는 우선 순위 역전 현상에 대해 그 이유와 배경을 예를 들어 알아보자. 예를 들어 인터넷에 직접 접속되어 있는 카메라가 포착한 화면을 인터넷을 통해 사용자에게 전송하는 인터넷 디바이스 웹 카메라를 고려해 보자. 사용자는 웹 브라우저를 통해 해당 웹 카메라가 포착한 화면을 볼 수 있을 뿐만 아니라 카메라의 방향을 바꾸거나 초점을 조절하는 등의 제어를 수행할 수 있다. 웹 카메라가 침입 탐지용으로 사용되는 경우에, 웹 카메라는 포착한 화면을 주기적으로 전송하는 기능 이외에도 관리자가 긴급히 카메라의 방향전환을 요청하면 이미 취득한 화면의 전송 작업에 우선하여 관리자가 요청하는 곳으로 초점을 이동할 수 있어야 한다.

설명을 위하여, 침입 감지용 웹 카메라는 기본적으로 관리자의 요청에 의해 카메라 방향을 바꾸는 카메라 방향전환 태스크를 T_1, 주기적으로 화면의 내용을 캡쳐하는 태스크를 T_2, 그리고 웹으로부터의 요구를 분석하고, 요구에 따라 해당 웹 문서나 CGI 프로그램을 실행시키는 웹서버를 T_3라 하자. 침입의 탐지를 위한 카메라의 방향전환 요청이 다른 작업에 앞서 우선적으로 처리되지 않는 경우 침입 탐지에 실패할 수 있다고 보고 T_1의 우선 순위가 다른 두 태스크 T_2과 T_3보다 높다고 가정하자. 만약, 실시간 내장형 운영체제와 그 위에서 실행되는 웹서버를 내장한 웹 카메라에서 침입이 탐지되어 관리자가 긴급히 카메라의 방향전환을 요청하는 경우 다음과 같은 경로를 따라 카메라 초점 이동 작업을 수행하는 태스크 T_1이 실행된다.

화면 캡쳐 태스크인 T_2가 실행 중 일 때 방향 전환 요청이 디바이스에 도착하면, 커널에 패킷 인터럽트가 발생한다. 이 때 RTOS 커널은 태스크 T_2를 잠시 중지하고 인터럽트 처리루틴을 실행하여 도착된 패킷들을 버퍼링한 후 다음 실행할 태스크를 스케줄한다. RTOS 커널은 태스크 T_2의 우선 순위가 웹서버 태스크인 T3보다 높기 때문에 T_2를 먼저 스케줄링한다. 따라서 중지되었던 태스크 T_2가 다시 실행되어 화면을 캡쳐한다.

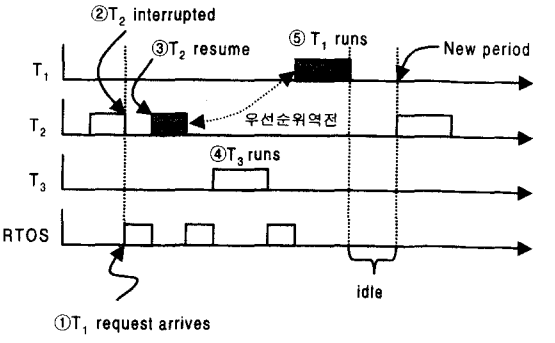


그림 1. 웹서버의 우선 순위가 낮은 경우 우선 순위 역전 현상

Fig. 1. Priority inversion in web server with low priority.

태스크 T_2가 종료된 후 T_3가 실행되고, 그 다음 T_1이 실행된다. 이는 RTOS가 방향 전환 요청이 태스크 T_1과 관련된다는 사실을 알더라도 패킷을 분석하기 전까지는 자신이 수신한 패킷에 그러한 요청이 있음을 알 수 없기 때문이다. 따라서 우선 순위가 높은 T_1 태스크보다 우선 순위가 낮은 T_2가 먼저 실행되는 우선 순위 역전 현상이 발생하며 이를 그림으로 표현하면 <그림 1>과 같다.

이처럼 모든 웹 요청을 단순 태스크로 작동되는 웹서버가 수신하고 분석한 후에 해당 작업이 이루어지는 경우에는 웹서버도 운영체제 위에서 실행되는 하나의 태스크 작업에 불과하기 때문에 아무리 높은 우선 순위의 작업이 인터넷을 통해 도착하더라도 웹서버가 동작한 후에 동작된다. 따라서 작업의 실시간 속성을 만족시켜주지 못하는 상황이 발생하고 우선 순위 역전 현상도 발생한다.

이를 해결하는 방법의 하나로 웹서버 태스크 T_3의 우선 순위를 타 태스크인 T_2보다 높게 할 수 있다. 이 경우 T_1이 T_2보다 먼저 스케줄 된다. 그러나 웹서버 T_3는 내부적으로 패킷 분석 기능뿐만 아니라 웹 문서를 읽어서 전송하는 암시적인 태스크 기능을 포함하고 있다. 따라서 T_3의 우선 순위를 다른 태스크보다 높게 할 경우 웹 문서를 읽어서 전송하는 암시적인 태스크의 우선 순위까지 높게 하는 결과를 가져온다. 결국 단순한 웹 문서를 요청하는 경우에도 웹 문서를 읽어서 전송하는 작업이 다른 태스크보다 먼저 수행되는 우선 순위 역적 현상이 발생한다. <그림 2>는 T_2가 실행 중일 때 단순한 웹 문서 요청을 처리하는 과정에서 발생하는 우선 순위 역전 현상을 보여준다.

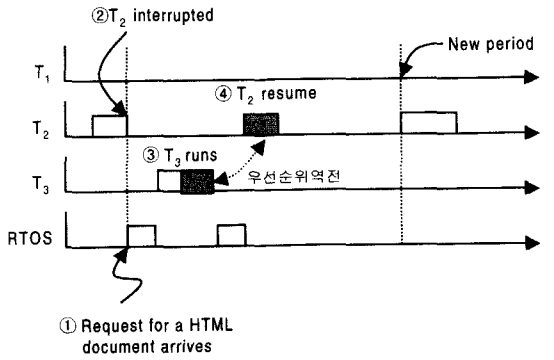


그림 2. 웹서버의 우선 순위가 높은 경우 우선 순위 역전 현상

Fig. 2. Priority inversion in web server with high priority.

이와 같이 웹서버를 하나의 태스크로 구현하는 경우에는 웹서버 태스크의 우선 순위에 관계없이 우선 순위 역전 현상이 발생함을 볼 수 있다. 우선 순위 역전 현상은 결과적으로 인터넷 디바이스가 실시간 스케줄링을 보장하지 못하는 결과를 초래할 수 있다. 이러한 우선 순위 역전 현상이 발생하는 이유는 스케줄링이 이원화되어 있기 때문이다. 이러한 문제점은 이원화된 스케줄링을 커널에 통합함으로써 웹 요청에 의해 실행되는 웹 태스크와 디바이스 내부 이벤트에 따라 실행되는 내부 태스크들을 일관된 방법으로 스케줄하여 해결할 수 있다.

Ⅲ. 통합 스케줄링

1. 통합스케줄러 구조

이원화된 스케줄링을 커널 내부에 통합하는 방법은 다음과 같다. 웹서버의 기능 중에서 스케줄링에 필수적인 기능, 즉 웹으로부터의 요구를 분석하고(Parsing), 실행할 웹 태스크(CGI)를 결정하여 태스크 우선 순위를 부여하는 기능을 웹서버로부터 분리하여 패킷분석기(Packet Analyzer)로 구성하여 실시간 커널 내에 존재하도록 하였다. 이러한 방법으로 구성된 내장형 실시간 웹서버의 구조는 <그림 3>과 같다. 웹으로부터 요청을 받아 패킷을 분석하고 스케줄링하는 커널 부분과 실제 웹 요청을 처리하고 디바이스 내부의 제어와 감시를 실행하는 태스크 부분으로 구분된다.

패킷분석기는 인터넷을 통한 웹 요청을 커널 내에서 분석하고, 요청된 URL(Uniform Resource Locator)에 따라 패킷 필터 데이타베이스(Packet Filter Database)를 참조하여 실행해야 할 웹 태스크와 우선 순위를 결정하고 스케줄러를 호출한다. 그리고 웹 요청을 해당 태스크의 소켓 포트로 전송(Redirection)한다. 패킷 필터 데이타베이스는 웹 브라우저로부터 요청된 URL, URL에 해당하는 이벤트 식별자, 이벤트 발생시 수행되어야 할 태스크 식별자, 태스크 우선 순위 그리고 태스크 소켓 포트번호 등의 정보를 관리하며 커널 내에 저장된다. 스케줄러는 패킷에 명시된 태스크(CGI)가 휴면 상태일 경우에는 준비(Ready) 상태로 전환한 후, 각 태스크의 우선 순위에 따라 스케줄하고 스케줄 된 태스크가 실행되도록 한다.

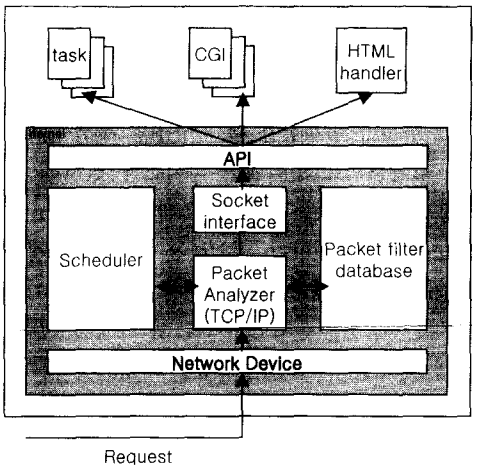


그림 3. 실시간 내장형 웹서버 구조
Fig. 3. Real-time embedded web server structure.

그 밖의 기능, 예를 들면 웹 브라우저로의 출력과 같이 태스크의 우선 순위와 관계없는 기능들은 실시간 운영체제 위에 HTML 핸들러 태스크로 운영되도록 하였다. 또한 웹 요청을 처리하는 웹 태스크와 디바이스의 제어나 감시기능을 수행하는 태스크가 태스크 부분에서 실행된다. 따라서 웹 브라우저로부터 웹 태스크 실행을 요청하는 패킷이 도착하는 경우, 스케줄러에 의해 우선 순위에 따라 웹 태스크가 직접 실행된다.

웹 태스크는 웹 브라우저로부터 PUT 또는 POST 형식으로 요청된다. 웹 태스크는 이벤트 식별자, 파라메터 규칙, 공통 자료구조들이 표준화된 형태로 제작되고 실시간 스케줄링을 위한 각종 속성(실행시간, 주기, 마감시간 등)과 함께 패킷 필터 데이타베이스에 등록된다. 내부 태스크는 시스템 설계시 지정한 이벤트 식별자, 스케줄링 속성 정보 그리고 메시지 교환을 위한 공통 자료구조에 따라 제작된다.

<그림 4>는 2장에서 예를 들었던 웹 카메라에서 방향전환 요청이 도착한 경우에 우선 순위가 가장 큰 태스크 T_1이 새로운 구조상에서 실행될 때까지의 과정을 보여주고 있다. 이 그림에서 보듯이 방향 전환 태스크 T_1이 T_2보다 먼저 우선적으로 처리됨을 알 수 있다.

이러한 방법은 우선 순위가 높은 태스크가 실행중일 때, 우선 순위가 낮은 태스크의 실행을 요구하는 패킷이 도착하면 우선 순위가 높은 태스크가 잠시 중단되는 현상이 발생할 수 있다. 이 때 통합 스케줄러는 웹 태스크 판별을 위하여 패킷을 분석하는 작업을 수행하기 때문에 스케줄링의 통합에 따른 추가적인 부담이

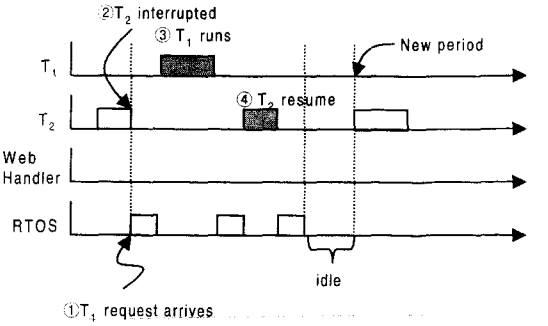


그림 4. 새로운 구조에서의 태스크 T_1 처리 과정
Fig. 4. Execution of task T_1 under new structure.

발생한다. 그러나 일반적으로 패킷이 네트워크 장치로 도착하는 경우에 운영체제는 다른 태스크의 실행을 잠시 중지하고 이를 처리하기 때문에 패킷의 내용을 간단히 분석하는 작업만이 추가로 요구되어 큰 부담으로 보기는 어렵다.

2. 패킷분석기의 비주기성 및 가변성 문제 해결

패킷의 내용을 간략히 분석하는 작업만 추가한 패킷 분석기의 경우에도 이를 커널에 둘 경우에는 웹 요청이 발생할 때마다 실행중인 태스크의 실행을 중지하고 패킷분석기의 기능을 수행해야 하므로 웹 요청이 집중된다면 우선 순위가 높은 태스크의 실행에 중대한 부담으로 작용할 수 있다. 결국 실시간성을 보장하지 못하는 결과를 초래할 수 있다. 이는 근본적으로 패킷분석기가 커널 내에서 인터럽트 유도형(Interrupt-Driven) 방식으로 동작하기 때문이다.

이 문제는 패킷분석기를 비주기적 태스크로 전환하여 대역폭 보존 서버(Bandwidth Preserving Server)[11] 기법으로 해결 가능할 것으로 본다. 이 기법은 일정한 크기의 프로세서 이용률을 가진 태스크(서버)를 두고 이 태스크로 하여금 비주기적인 서비스를 하게 하는 방법이다. 서버는 우선 순위 유도형 스케줄링 알고리즘에 의해 지원되며, 주기적 태스크와 유사한 방법으로 스케줄링 된다. 서버는 수행할 비주기적 작업(패킷 분석 작업)이 도착해 있고, 실행 예산이 남아 있을 경우에만 수행된다. 이러한 서버는 실행 예산을 언제 얼마만큼 감소할 것인지를 정의한 소비 규칙(Consumption Rules)과 언제 얼마만큼을 증가할 것인지를 정의한 보충 규칙(Replenishment Rules)에 따라 Deferrable Server(DS)[8], Sporadic Server(SS), Constant Utilization Server(CUS), Total Bandwidth Server(TBS),

Constant Bandwidth Server(CBS) 등 다양한 종류가 있다. DS와 SS는 고정 우선 순위 스케줄링을 기반으로 하면서 주기적 태스크처럼 스케줄링 되는 반면, TBS, CUS, CBS는 동적 우선 순위 스케줄링을 기반으로 스포래딕(Sporadic) 태스크처럼 스케줄 된다. 이러한 패킷 분석기의 가변성 문제는 본 논문에서 해결하고자 하는 문제와 다른 문제이기 때문에 구체적인 해결 방안에 대해서는 더 이상 언급하지 않는다.

Ⅳ. 실시간 서비스 인터넷 디바이스를 위한 경량 웹서버에의 응용

본 논문에서 제안한 내장형 웹서버 모델의 구현 가능성, 성능 분석 및 연구 분석을 위하여 ARM 마이크로프로세서와 이더넷 어댑터를 내장한 전용 보드를 개발하였다. 전용 보드는 16MB 메모리와 인터넷 접속을 위한 RJ-45 이더넷 입력포트와 콘솔용 직렬 포트를 가지고 있다. <그림 5>는 전용보드의 하드웨어 블록도와 전용보드 위에 탑재된 소프트웨어 구성을 보여주고 있다.

운영체제는 μC/OS[9]을 사용하였다. μC/OS는 소스 코드가 공개된 실험용 실시간 운영체제로서 여러 연구에서 사용되고 있다. 태스크의 스케줄을 위하여 고정 우선 순위 방식의 스케줄링 알고리즘을 사용한다. 비주기적 태스크의 경우, 본 논문에서는 실시간성을 요구하는 높은 우선 순위 비주기적 태스크의 응답 시간을 향상시키기 위하여 대역폭 보존 서버 기법을 적용하지 않고 인터럽트 유도형 방식으로 스케줄하도록 하였다. TCP/IP는 Waterloo TCP/IP[10]를 사용하였다. Waterloo TCP/IP는 소스 코드가 공개되어 있는 경량 TCP/IP 커널이다. 실험 환경을 위하여 Waterloo TCP/IP에서 Telnet 서버, FTP 클라이언트 등을 제거하고 웹 서비스에 필요한 최소한의 프로토콜 규약만을 포함하도록 소스 코드를 수정하였으며, 실시간 커널 내에 구현한 것과 동일한 효과를 갖기 위하여 μC/OS 운영체제상의 최상위 우선 순위 태스크로 구동하였다

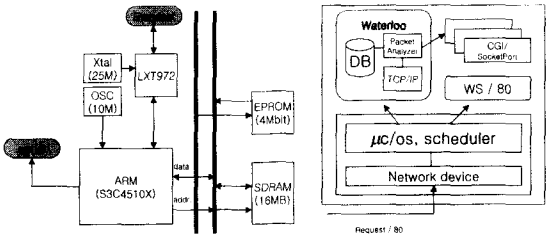


그림 5. 실험용 하드웨어 블록도와 소프트웨어 구성도
Fig. 5. Experimental hardware and software block diagram.

패킷분석기는 Waterloo TCP/IP에 패킷 분석 기능을 추가하여 구현하였다. 패킷분석기의 기본적인 기능은 <그림 6>과 같다. 패킷분석기는 웹 요청 여부를 확인하기 위하여 80번 소켓 포트를 지속적으로 감시한다. 해당 포트에 패킷이 도착하면 패킷분석기는 실행할 웹 태스크와 실행 우선 순위 및 소켓포트번호 등의 스케줄링 정보를 <그림 7>과 같은 패킷 필터 데이타베이스로부터 가져와 해당 요청을 처리할 태스크를 결정한다. 그런데 실시간 웹 서비스가 아닌 인터넷 요청 또는 일반 웹 문서를 요청하는 경우에도 커널에서 패킷을 분석한다. 따라서 웹을 통한 실시간 작업 요청이 아닌 경우에는 패킷분석기가 웹서버와 운영체제 스케줄링을 통합함으로써 발생하는 부담으로 작용하게 된다.

고정된 상수의 디스패치(Dispatch) 시간을 보장하기 위해 μC/OS에서 사용된 다단계 비트 벡터(Multi-layered Bit Vector)와 비트 위치 변환 테이블을 활용하였다. 이 기법은 최우선 순위를 찾기 위하여 별도의 비트 벡터 및 변환 테이블을 필요로 하지만 작업 수에 상관없는 고정된 디스패치 시간을 보장하는 장점을 제공한다.

웹 태스크 제작시 패킷 필터 데이타베이스에 웹 태스크 정보를 등록하기 위하여 <표 1>과 같은 함수 인터페이스를 제작하였다.

```
Packet_Analyzer(client, URL)
{
   if(Pattern(URL) == CGI) { /* 웹 태스크 실행 요청 */
   TaskInfo = FindPacketFilterDB(URL);
   if(TaskInfo != NULL){
   Schedule(TaskInfo);
   RedirectHTTP(client, TaskInfo.socketPort);
   } else
   ReplyError(client, "Bad Request");
   }
   else { /* 일반 웹 문서 요청 */
   Schedule(HTMLHandler);
   RedirectHTTP(client, HTMLHandler.socketPort);
   }
}
```

그림 6. 패킷분석기의 기능
Fig. 6. Packet analyzer function.

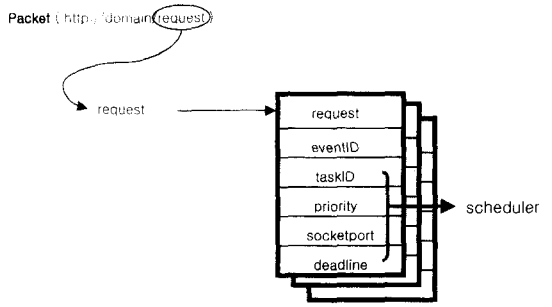


그림 7. 패킷 필터 데이터베이스
Fig. 7. Packet filter database.

표 1. 웹 태스크 등록 함수 인터페이스
Table 1. Function Interfaces to register web task.

자료형	이 름	설 명
eventId	registerEvent(request);	태스크가 처리할 웹 요청 등록
int	registerTask(eventId, sockerport, priority, deadline);	태스크 스케줄링 정보 등록

registerEvent()는 해당 태스크가 웹으로부터 어떤 요청에 대하여 처리할 것인지를 패킷 필터 데이타베이스에 등록한다. 해당 요청에 대한 처리 태스크가 이미 다른 태스크에 의해 패킷 필터 데이타베이스에 등록되어 있지 않을 경우는 웹 요청과 해당 태스크의 식별자를 등록한다. registerTask()는 등록한 태스크의 스케줄에 필요한 정보와 웹 요청 내용을 주고받을 소켓번호를 등록한다.

웹서버를 구성하는 CGI 태스크, HTML 핸들러 태스크들은 개별 함수로 코딩된 후 μC/OS와 함께 링크되어 단일 실행 가능한 프로그램으로 생성된 후 전용 보드의 부팅 후 모든 작업을 처리한다. HTML 핸들러 태스크는 소켓으로부터 웹 문서 요청을 대기하고 있는 태스크이다. 홈페이지 저장을 위하여 μC/OS 운영체제 상에서 별도의 파일시스템을 구현하지 않고 대신 시스템이 차지하는 메모리 공간의 일부를 가상 파일시스템으로 사용하였다. 이를 위하여 웹서버가 호출하여 사용할 수 있는 가상 파일시스템 API를 구현하였다. 가상 파일시스템은 기본적으로 파일경로 이동, 파일 열기, 파일 읽기, 파일 닫기의 기능을 제공하며, 가상 파일시스템 생성을 위하여 별도의 유틸리티를 사용한다. 이것은 파일을 링크 가능한 형태의 C언어 프로그램으로 변환하고 μC/OS와 함께 링크함으로써 가능하다.

V. 성능평가

제안된 구조의 성능을 확인하기 위하여 <그림 8>에서와 같이 패킷 분석 기능을 커널 내에 두는 새로운 구조와 웹서버 내에 두는 기존 구조에서 수행 성능을 비교하는 실험을 수행하였다.

실험 환경을 위해 외부 침입 감지시 관리자의 요청에 의해서만 동작하는 최우선 실행 우선 순위의 카메라 방향전환 태스크 T_3와 화면을 캡처하거나 전송하는 등의 주기적으로 동작하는 시스템 내부 태스크 T_2를 두었다. 기존 구조에서 태스크 WS는 일반적인 웹서버 역할을 수행하며, 새로운 구조에서는 HTML 핸들러 태스크의 역할을 수행하도록 하였다.

시스템 내부 부하 증가에 따라 최우선 순위 태스크 T_3에 끼치는 영향을 관찰하기 위해서 태스크 T_2수의 증가를 통해 시스템 내부 부하를 가중하였다. 태스크 T_2는 실행 시간이 10ms이며 1000ms의 주기로 반복적으로 수행되는 시스템 내부 태스크로 제작하였다. 따라서 T_2를 최대 50개까지 실행시키는 경우 시스템에 가해지는 부하는 50%이다.

성능 척도로는 높은 우선 순위의 태스크 T_3의 응답 시간을 사용하였다. 웹 브라우저를 통하여 태스크 T_3 실행을 요청하는 경우, 네트워크 환경의 가변적 요소를 배제하기 위하여 웹 요청 도착 시각부터 태스크 T_3의 실행 종료 시각까지의 경과 시간을 응답 시간으로 이용하였다. R321은 기존 구조에서 우선 순위를 $T_3 > T_2 > WS$와 같이 웹서버를 낮은 우선 순위로 설정한 일반적인 경우이다. R311은 새로운 구조에서 우선 순위를 R321과 동일하게 설정한 경우이다. R322는 기존 구조에서 우선 순위를 $WS > T_3 > T_2$와 같이 웹서버를 가장 높은 우선 순위로 설정한 경우이다. 결과는 <그림 9>와 같다.

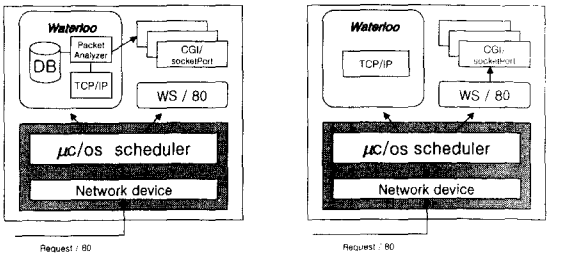


그림 8. 새로운 구조와 기존 구조
Fig. 8. Proposed structure and traditional structure.

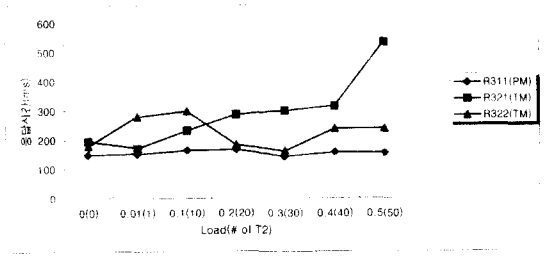


그림 9. 태스크 T_3의 응답 시간(클라이언트 수 : 1)
Fig. 9. Response time of task T_3(Number of clients : 1).

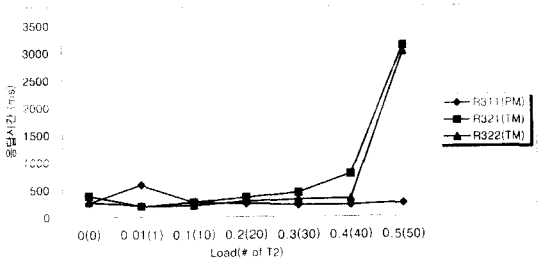


그림 10. 태스크 T_3의 응답 시간(클라이언트 수 : 4)
Fig. 10. Response time of task T_3(Number of clients : 4).

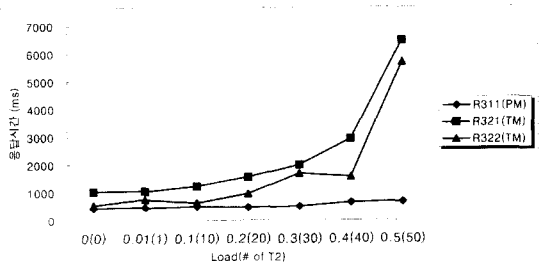


그림 11. 태스크 T_3의 응답 시간(클라이언트 수 : 8)
Fig. 11. Response time of task T_3(Number of clients : 8).

기존 구조인 R321에서는 태스크 T_2의 수가 늘어나 시스템 부하가 증가하는 경우에 최상위 우선 순위 태스크 T_3의 응답 시간도 비례하여 증가됨을 알 수 있다. 이것은 웹서버가 패킷 분석을 마친 후에야 비로소 태스크 T_3가 수행되기 때문이다. 즉 태스크 T_2의 수가 증가함에 따라 T_2의 우선 순위가 웹서버보다 높기 때문에 웹서버의 실행을 지연시키고 있는 것이다. 그러나 R311의 응답 시간을 보면, 새로운 구조에서는 부하가 적은 경우에 기존 구조와 유사한 성능을 보여주고 있으며 시스템 부하가 증가하는 경우에도 응답 시간이 일정하여 기존 구조보다 향상된 성능을 보여준다. 이것은 패킷 분석이 커널 내에서 이루어져 우선 순위 역전 현상이 발생하지 않기 때문이다.

그런데 기존 구조에서 웹서버의 실행이 지연되는 현상을 방지하기 위하여 웹서버의 우선 순위를 WS > T_3 > T_2와 같이 가장 높게 설정할 수 있다. R322는 이와 같이 기존 구조에서 웹서버 우선 순위를 가장 높게 설정하여 실험한 결과이다. R322에서는 시스템 부하가 증가하여도 응답 시간이 증가하지 않아 새로운 구조에서와 유사한 결과를 보여주고 있다. 이것은 태스크 T_2수가 증가하더라도 웹서버가 우선하여 실행되기 때문이다. 그러나 이 경우는 우선 순위가 낮은 태스크를 요청하는 경우에도 항상 웹서버가 우선하여 실행되게 되어 결과적으로 부담으로 작용하게 된다.

<그림 10>과 <그림 11>은 웹서버로 동시에 요청하는 클라이언트 수가 각각 4개와 8개인 경우 응답 시간을 측정한 결과이다. 웹서버의 부하가 증가하기 때문에 전체적으로 응답 시간이 증가하였다. 이 경우에도 T_2의 개수가 작을 때는 기존 구조와 제안 구조가 유사한 성능을 나타내고 있으나 T_2수가 증가함에 따라 기존 구조의 응답 시간(R321, R322)이 증가하고 있다. 이것은 T_2수가 증가함에 따라 우선 순위 역상 현상이 더욱 심각해지고 있음을 보여주고 있는 것이다. 그런데 기존 구조에서 T_2의 부하가 0.4이후에 응답 시간이 급격히 증가하는 현상을 보여주고 있다. 이것은 기존 구조에서 클라이언트 수가 증가함에 따라 웹서버가 계속 실행되어야 하는 부담의 증가와 커널 내부 부하 그리고 빈번한 T_2의 마감시간 초과(Deadline Miss) 현상으로 인해 T_3의 실행을 지연시키는 때문이다. 즉 기존 구조(R321, R322)에서는 웹 태스크를 요청하는 경우에 웹서버 태스크가 실제로 실행되지만 제안구조(R311)에서는 커널에서 직접 스케줄하기 때문에 웹서버(HTML Handler)는 실제로 실행되지 않는다. 또한 T_2의 부하가 0.5인 경우 이미 시스템은 부하가 매우 높아져 마감시간 초과 현상 많이 발생한다. 각 환경에서 부하에 따라 발생하는 마감시간 초과 현상에 대한 실험은 아래에서 설명한다.

<그림 12>, <그림 13> 그리고 <그림 14>는 각 환경에서 클라이언트 수에 따른 T_3 응답시간 변화를 보여주고 있다. 클라이언트 수의 증가는 T_2수의 증가와 마찬가지로 웹서버 부하에 영향을 미친다. <그림 13>은 기존 구조에서는 클라이언트 수의 증가에 따른 응답 시간 증가 비율이 시스템 부하의 증가비율보다 더

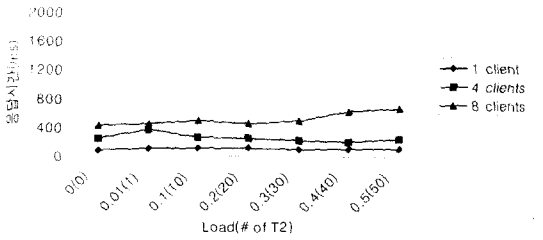


그림 12. 클라이언트 수에 따른 태스크 T_3의 응답 시간(제안 구조, R311)

Fig. 12. Response time of task T_3 as number of clients increase(Proposed Model, R311).

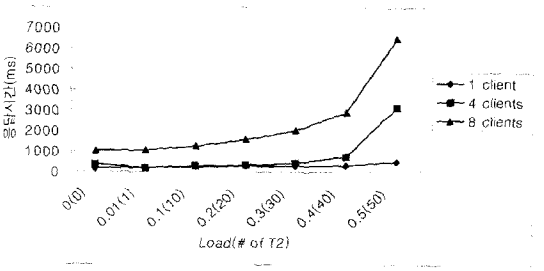


그림 13. 클라이언트 수에 따른 태스크 T_3의 응답 시간(기존 구조, R321)

Fig. 13. Response time of task T_3 as number of clients increase(Traditional Model, R321).

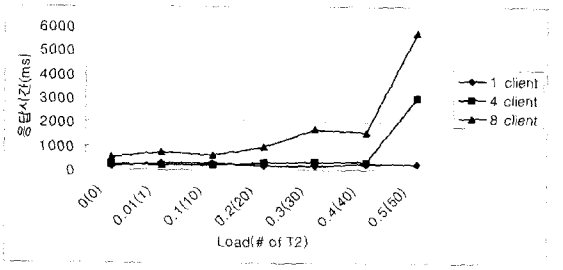


그림 14. 클라이언트 수에 따른 태스크 T_3의 응답 시간(기존 구조, R322)

Fig. 14. Response time of task T_3 as number of clients increase(Traditional Model, R322).

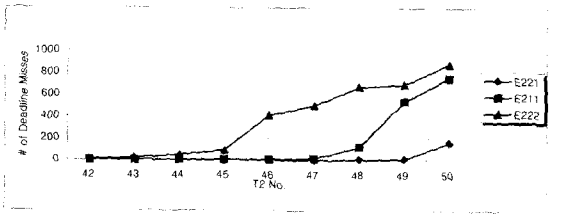


그림 12. 태스크 T_2의 마감시간 초과 수

Fig. 12. Number of deadline misses of task T_2.

커지고 있음을 알 수 있다. 이것은 클라이언트 수와 부하가 증가함에 따라 우선 순위 역전 현상이 끼치는 영향이 더욱 커지기 때문이다. <그림 14>에서는 웹서버가 가장 높은 태스크이기 때문에 부하가 일정 수준까지 증가하여도 응답 시간은 크게 증가하지 않고 있다. 이것은 T_2와 웹서버간의 우선 순위 역전 현상이 R321과 같이 심각하게 발생하지 않기 때문이다. 그러나 앞에서 언급한 것과 같이 낮은 우선 순위의 웹 요청이 발생하는 경우에도 항상 웹서버가 항상 실행되므로 높은 우선 순위 태스크의 실행을 지연시킬 수 있는 문제점을 가지고 있다. 제안된 구조에서는 <그림 12>가 나타내는 바와 같이 클라이언트 수 증가하는 경우에도 시스템 부하에 관계없이 응답 시간의 증가가 기존 구조에 비하여 일정함을 알 수 있다.

새로운 구조에서 패킷분석기를 커널 내에 둠으로써 생기는 추가적인 부하와 기존 구조에서 웹서버로 인해 발생하는 부하를 비교 측정하였다. 이를 위하여 새로운 태스크 T_0를 추가하였다. 태스크 T_0는 웹 요청에 의해 단순히 시스템 상태를 보여주는 작업이며 가장 낮은 우선 순위로 지정하였다. 부하를 측정하기 위하여 태스크 T_0를 반복적으로 요청할 때 발생하는 태스크 T_2의 마감시간 초과 횟수를 척도로 사용하였다. <그림 12>는 50개의 태스크 T_2가 실행중인 상태에서 태스크 T_0의 실행을 1000번 요청하는 경우 각각의 태스크 T_2에서 발생한 마감시간 초과 횟수를 나타낸다. E211과 E221은 각각 새로운 구조와 기존 구조에서 우선 순위를 $T_3 > T_2 > WS > T_0$으로 동일하게 설정한 경우이다. 그리고 E222는 기존 구조에서 우선 순위를 $WS > T_3 > T_2 > T_0$와 같이 웹서버를 최상위 우선 순위로 설정한 경우이다.

E222 환경에서 가장 많은 마감시간 초과 현상이 발생하였다. 이것은 웹서버가 가장 우선 순위가 높기 때문에 태스크 T_2의 실행을 지연시킨 결과이다. E211 환경에서는 패킷분석기를 커널 내에 둠으로써 생기는 부담으로 인하여 E221보다 마감시간 초과 현상이 증가하였다. 그러나 E222보다는 마감시간 초과 현상이 적게 발생하였다. 실험 결과를 다른 관점에서 해석한다면, 웹서버를 낮은 우선 순위로 하는 기존 구조에서는 47개의 T_2가 스케줄 가능하고, 새로운 구조에서는 스케줄 가능한 T_2의 수가 44개로 감소하였다. 이것은 패킷 분석 기능을 커널 내에 둠으로써 생기는 부하로 인해 발생한다고 볼 수 있다. 그러나 기존 구조에서 웹서버를

최상위로 하는 경우에는 스케줄 가능한 T_2의 수는 41개로 더욱 감소하였다. 이것은 기존 구조에서 웹서버를 최상위로 하는 경우에 생기는 부하가 패킷 분석 기능을 커널 내에 두는 경우보다 상대적으로 크다는 것을 의미한다.

결과적으로 새로운 구조는 웹서버를 낮은 우선 순위로 설정하더라도 실시간 태스크의 응답 시간이 일정하여 기존 구조에서 웹서버가 최우선 순위로 동작하는 것과 유사한 성능을 보여줄 뿐 만 아니라 부하 측면에서도 패킷 분석 기능의 추가로 인해 발생하는 부하가 기존 구조에서 웹서버를 최우선 순위로 설정하는 경우보다 작다. 따라서 웹을 통한 작업 요청이 실시간성을 보장해야 하는 경우, 패킷 분석 기능을 커널에 통합하여 스케줄링하는 새로운 구조가 웹서버 자체를 최상위 우선 순위로 지정하는 것보다 더 적합함을 확인하였다.

Ⅵ. 결론

본 논문에서는 인터넷 디바이스에 웹서버가 탑재되는 경우 실시간 운영체제와 웹서버간의 이원화된 스케줄링 때문에 발생하는 우선 순위 역전 현상을 해결하기 위하여 웹서버의 일부 스케줄링 기능을 운영체제 커널 내에 포함시켜 이원화된 스케줄링을 통합하는 기법을 제안하였다. 제안된 기법은 웹서버의 패킷 분석 기능을 분리하여 커널 내에 구현하게 되므로 인터넷을 통하여 접속이 가능한 인터넷 디바이스에 쉽게 적용 가능하다.

제안된 기법의 성능을 실험하기 위하여 프로토타입 내장형 시스템을 제작하였으며, 이를 통한 실험 결과, 웹서버를 일반 태스크로 구현한 기존의 내장형 시스템과 달리 우선 순위 역전 현상이 발생하지 않아 실시간 성질을 보장해야 하는 태스크의 응답 시간이 시스템 부하에 관계없이 일정함을 보여주었다. 또한 마감시간 초과 실험을 통하여 제안된 기법에서 스케줄링을 통합함으로써 발생하는 부하가 웹서버를 최상위 우선 순위로 동작시키는 기존 구조보다 적다는 것을 간접적으로 보여주었다. 인터넷을 통하여 접속이 가능하면서 태스크의 실시간성이 보장되어야 하는 인터넷 디바이스의 실시간 스케줄링에 제안된 기법이 적용될 수 있을 것이다.

향후 연구에서는 통합 스케줄링 환경에서 웹 요청이 빈번할 경우 발생하는 비주기적 태스크의 가변성 문제를 해결하는 방안과 스케줄링 통합에 따른 부하에 대하여 다양한 실험을 통하여 자세히 분석하고자 한다.

참 고 문 헌

[1] Hong-Taek Ju, Mi-Joung Choi and James W. Hong "An efficient and lightweight embedded Web server for Web-based network element management," International Journal of Network Management, vol. 10, Issue 5, pp. 261~275, September/October 2000.

[2] Jae Chul Moon, Soon Ju Kang, "An Event Channel-Based Embedded Software Architecture for Developing Telemetric and Teleoperation Systems on the WWW," IEEE Int'l Symposium on Real-Time Technology and Application(RTAS99), Vancouver, CANADA, June 2-4 1999.

[3] Arjan van de Ven, khttpd, http://www.fenrus.demon.nl, 1999.

[4] Zach Brown, phttpd, http://www.zabbo.net/phhttpd, November 1999.

[5] Chuck Lever, Marius Asmodt Eriksen and Stephen P. Molloy, "An analysis of the TUX web server," CITI technical Report 00-8, November 16, 2000.

[6] Jussara Almeida, Mihaela Dabu, Anand Manikutty and Pei Cao, "Providing Differentiated Levels of Service in Web Content Hosting," in First Workshop on Internet Server Performance, Madison, Wisconsin, June 23 1998.

[7] M. E. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection Scheduling in Web Servers," in Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems (USITS '99), Boulder, Colorado, October 1999.

[8] J. P. Lehoczky, L. Sha, and J. K. Stronider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," Proc of Real-Time Systems Symposium, pp. 261~270, Dec 1987.

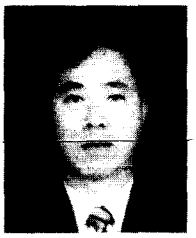
[9] Jean J. Labrosse, MicroC/OS-II, R&D Books,

1999.

[10] Erick Engelke, WATTCP, http://www.wattcp.com, 1999.

[11] Jane W. S. Liu, Real-Time Systems, Prentice Hall, 2000.

저 자 소 개

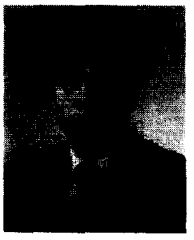


姜 奉 直(正會員)

1989년 : 서울대학교 계산통계학과 학사. 1991년 : 한국과학기술원 전산학과 석사. 2000년 : 아주대학교 컴퓨터공학과 박사수료. 1991년~1994년 : 포스데이타 근무. 1995년~현재 : 동양공업전문대학 전산경영기술공학부 재직. <주관심분야 : 운영체제, 실시간 시스템, 내장형 시스템>

李 賢 淑(正會員)

2001년 : 아주대학교 정보 및 컴퓨터공학부 학사. 현재 : 아주대학교 정보통신전문대학원 석사 4학기. <주관심분야 : 운영체제, 분산 시스템, 실시간 시스템>



鄭 己 鉉(正會員)

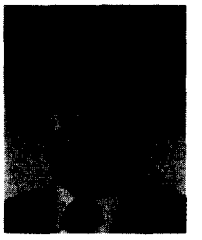
1984년 : 서강대학교 공과대학 전자공학과 학사. 1988년 : University of Illinois, EECS 석사. 1990년 : Purdue University 박사. 1984년~1986년 : FACOM Korea 근무. 1991년~1992년 : 현대전자 반도체 연구소 근무. 현재 : 아주대학교 전자공학부 재직. <주관심분야 : 컴퓨터구조, VLSI 설계, 멀티미디어 및 실시간 시스템 등>

丁 錫 龍(正會員)

1987년 : 서울대학교 계산통계학과 학사. 1993년 : 한국과학기술원 전산학과 석사. 2000년 : 아주대학교 컴퓨터공학과 박사수료. 1987년~1995년 : LG정보통신 근무. 1996년~현재 : 동양공업전문대학 전산경영기술공학부 재직. <주관심분야 : 운영체제, 실시간 시스템, 내장형 시스템>

崔 景 熙(正會員)

1976년 : 서울대학교 사범대학 수학교육과 학사. 1979년 : 프랑스 그랑데꼴 ENSEEIHT 정보공학과 석사. 1982년 : 프랑스 Paul Sabatier 정보공학과 박사. 현재 : 아주대학교 정보 및 컴퓨터공학부 재직. <주관심분야 : 운영체제, 분산 시스템, 실시간 및 멀티미디어시스템 등>



柳 海 永(正會員)

1979년 : 단국대학교 수학과 졸업(이학사). 1982년 : 단국대학교 대학원 수학과 수료(이학석사). 1994년 : 아주대학교 대학원 컴퓨터공학과 수료(공학박사). 1983년~현재 : 단국대학교 정보컴퓨터학부 재직. <주관심분야 : 소프트웨어공학, 정보화 전략계획 수립 및 컨설팅, 웹 트래픽 튜닝 등>