인터넷 가전의 원격 감시 및 제어를 위한 임베디드 리눅스 웹 서버

임 성 락*

An Embedded Linux Web Server for Remote Monitoring & Controlling the Internet Appliance

Seong-Rak Rim*

Abstract

Since most of the conventional web servers have been designed to provide the general purpose and user's convenience as the primary goal, there is an overhead to apply them to the embedded system for remote monitoring and controlling the operation status of the Internet appliance. To cope with this overhead, an embedded Linux web server is suggested in this paper. The basic concept is to provide miniaturization and extendability by simplifying the requirements of web server as the services to the requests of web document for the user's menu and the execution of CGI program for monitoring and controling the Internet appliance, and satisfying only the indispensable requirements of HTTP which are necessary for the interface with the conventional web browsers. For the evaluation of its feasibility, each module has been implemented on Linux environment, and tested with the MS Explorer 6.0.

Keywords : Web Server, Embedded System

논문접수일 : 2005년 7월 5일 논문게재확정일 : 2005년 12월 5일

※ 본 논문은 2004년도 호서대학교 학술연구조성비에 의하여 연구되었음.

* 호서대학교 컴퓨터공학부 교수 (336-795)충남 아산시 배방면 세출리 산 29-1
Tel : 041-540-5708. Fax : 041-548-9667. e-mail : srrim@office.hoseo.ac.kr

1. 서 론

최근 출시되고 있는 디지털 TV를 포함하여 대부분의 인터넷 가전들은 텍스트는 물론 음성, 영상 등 다양한 종류의 정보를 처리하기 위하여 임베디드 시스템을 도입하고 있는 추세이다. 뿐만 아니라 자동 제어를 필요로 하는 장비를 포함하여 공장 자동화 장비에서도 원격 감시 및 제어를 위하여 임베디드 시스템이 적용되고 있다. 이러한 추세는 핸드폰 혹은 PDA와 같은 단말기를 통해 언제 어디서나 인터넷에 연결된 시스템들을 접근할 수 있는 유비쿼터스 환경에 대응하기 위하여 지속적으로 증가하고 있는 실정이다[이봉규외 1인, 2004 ; 임채덕, 2004].

인터넷 가전을 웹 브라우저를 통하여 원격 제어하기 위해서는 웹 서버가 요구된다. 웹 서버는 웹 브라우저를 통해 어떤 요청이 들어왔을 때 이를 처리하기 위한 소프트웨어 혹은 컴퓨터 시스템을 일컫는다[David et. al., 2002]. 한편 임베디드 소프트웨어는 PC와 같이 비교적 넉넉한 환경에서 실행되는 소프트웨어와 달리 제약된 환경하에서 시스템이 요구하는 기능만을 최적으로 제공하는 것을 대전제로 하고 있다[김두현외 1인, 2004]. 이러한 전제 하에서 볼 때, 인터넷 가전을 원격 감시 및 제어하기 위한 임베디드 웹 서버는 기본적으로 그 크기가 작고, 새로운 기능 추가를 위한 수정작업이 용이하여야 한다. 그러나 대부부의 기존 웹 서버들은 범용성과 사용자의 편의성을 위하여 HTTP 프로토콜의 요구사항을 완전히 만족시킬 뿐 아니라 다양한 기능과 향상된 성능을 제공하는 것을 주된 목적으로 설계되었기 때문에 그 크기가 너무 크고, 구조가 매우 복잡하여 이것들을 인터넷 가전의 동작 상태를 감시하고 제어하기 위한 임베디드 시스템에 적용하기에는 상당한 오버헤드가 있다.

일반적으로 임베디드 웹 서버의 주된 목적은 웹 브라우저를 통해 인터넷에 연결된 임베디드 시스템의 웹 문서를 읽을 수 있도록 하며, 이를 통해 임베디드 시스템에 연결된 장치들의 동작 상태를 원격으로 감시하고 제어할 수 있도록 한다 현재 가장 많이 사용되고 있는 임베디드 웹 서버로는 Boa[11]와 Goahead[12]가 있으며 이들 모두 버전 1.0 대이지만 많은 사용자들로부터 안정성을 인정받고 있다. 그러나 이들의 소스코드를 분석해 본 결과, 범용성을 위한 다양한 기능과 고성능을 제공하려는 지속적인 노력으로 그 크기가 방대해졌고, 그 구조가 너무 복잡하여 인터넷 가전의 감시 및 제어를 위한 임베디드 시스템에 적용할 경우 불필요한 기능까지도 감수해야 하며 수정작업은 거의 불가능함을 알 수 있었다. 이러한 어려움을 해결하기 위하여 본 논문에서는 인터넷 가전들의 동작 상태를 원격으로 감시 및 제어하기 위한 임베디드 시스템에 적합한 웹 서버를 제시하고 그 타당성을 검토한다.

2. 설 계

2.1 임베디드 웹 서버의 요구사항

임베디드 시스템과 범용 컴퓨터의 가장 큰 차이점은 그 목적에서 찾아 볼 수 있다. 범용 컴퓨터의 목적은 어떤 특정 문제만을 해결하기 위한 것이 아니지만, 임베디드 시스템은 어떤 특정 문제를 저렴한 비용으로 해결하는 것이 그 목적이다[John, 2001]. 또한 사용자 인터페이스에서 차이점을 발견할 수 있다. 일반적으로 범용 컴퓨터는 모니터, 키보드, 마우스와 같은 장치를 가지고 있다. 반면, 임베디드 시스템은 사용자 인터페이스를 위한 장치를 전혀 가지고 있지 않거나, 단순 버튼이나 터치 스크린 혹은 제어용 패널과 같은 특별한 인터페이스 장치를 가

지고 있다. 사용자 인터페이스 장치를 가지고 있지 않는 임베디드 시스템은 네트워크 혹은 센서를 통해 단순히 데이터를 수집하거나 명령을 받아 이를 처리한다.

이러한 관점에서 인터넷 가전의 동작상태 감시 및 제어를 위한 임베디드 웹 서버를 범용 웹 서버와 비교해 볼 때 다음과 같은 요구사항의 차이점이 있다.

2.1.1 입출력 구조

우선 이용자 측면에서 볼 때, 범용 웹 서버는 다수의 이용자들이 이용하게 되지만, 인터넷 가전을 위한 임베디드 웹 서버는 제한된 소수에게만 허락되어 이용하게 된다. 따라서 다중 접속 처리를 지원하기 위한 입출력 구조가 필수적으로 요구되는 것은 아니다.

2.1.2 HTTP 프로토콜

HTTP 프로토콜 측면에서 볼 때, 범용 웹 서버는 범용성을 위하여 가능한 한 모든 규약을 만족해야 하지만, 인터넷 가전을 위한 임베디드 웹 서버의 경우에는 감시 및 제어에 필요한 조건만 만족하면 된다. 예를 들어 임베디드 웹 서버는 웹 브라우저로부터 전송되는 정보의 양이 비교적 작아서 HTTP 규약의 다양한 Method 중에서 GET Method 만으로 처리하는 것이 메모리 관리에 효율적이다.

2.1.3 동적 웹 페이지

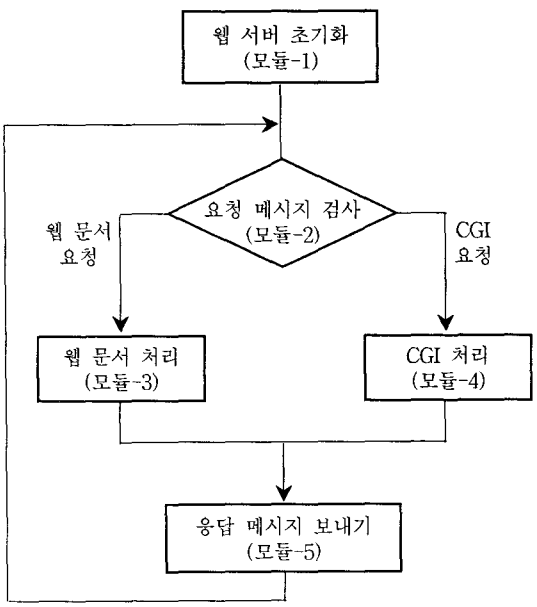
웹 서버는 정적, 동적 웹 페이지를 제공한다. 범용 웹 서버는 정보 제공을 위한 정적 웹 페이지를 주로 지원하지만, 인터넷 가전을 위한 임베디드 웹 서버의 경우에는 사용자 메뉴를 위한 정적 웹 페이지와 감시 및 제어를 위한 CGI 프로그램과 같은 동적 웹 페이지를 지원해야 한다 [강선례 외 1인, 2003].

2.1.4 이식성 및 확장성

범용 웹 서버 이용자는 웹 서버 자체 보다 다양한 정보 제공을 위한 웹 페이지에 대한 추가 및 수정 작업이 주로 이루어지지만, 인터넷 가전을 위한 임베디드 웹 서버의 경우에는 감시 및 제어하려는 가전제품에 따라 웹 페이지는 물론 웹 서버 자체에 대한 추가 및 수정 작업이 필요하다. 따라서 웹 서버의 이식성 및 확장성이 요구된다.

2.2 기본 개념

임베디드 소프트웨어는 하드웨어에 내장되기 때문에 소형화와 저렴한 가격이 필수적으로 요구된다. 또한 Time-to-Market이 생명인 임베디드 시스템 제조업체들은 꼭 필요한 기능만을 지원하는 임베디드 소프트웨어를 저렴한 비용으로 개발하기를 원하고 있다. 따라서 본 논문에는 임베디드 웹 서버의 소형화와 이식성 및 확장성 제공을 설계 목표로 한다.



〈그림 1〉 웹 서버 흐름도

인터넷 가전의 동작상태 감시 및 제어를 위한 임베디드 웹 서버의 요구사항을 고려해 볼 때 그 기능은 매우 단순하다. 즉, 사용자 메뉴를 위한 웹 문서 참조와 감시 및 제어를 위한 CGI 프로그램을 실행하는 것으로 정의할 수 있다. 본 논문에서 제시하는 임베디드 웹 서버의 전체 흐름도는 <그림 1>과 같으며 각각의 요소를 모듈화함으로써 이식성과 확장성을 제공하도록 한다.

2.3 모듈별 설계

2.3.1 모듈-1 : 웹 서버 초기화

웹 브라우저와 웹 서버는 클라이언트-서버 모델로서 본 논문에서는 상호간의 통신을 위해 선 리눅스에서 제공하는 TCP/IP 프로토콜을 이용하여 통신 채널을 생성한다.

```
 1 int sock_fd, new_sock, sock_len;
 2 struct sockaddr_in ser_addr, cli_addr;

/* 소켓 생성 */
 3 sock_fd = socket(AF_INET, SOCK_STREAM, 0);

/* IP 주소 및 포트 번호(80) 할당 */
 4 memset(&seraddr, 0, sizeof(seraddr));
 5 ser_addr,sin_family = AF_INET;
 6 ser_addr.sin_addr = htonl(INADD_ANY);
 7 ser_addr.sin_port = htons(80);
 8 bind(sock_fd, (struct sockaddr *)&ser_addr,
   sizeof(seraddr));

/* 연결요청 기다림 */
 9 listen(sock_fd, 1);

/* 연결요청 처리 */
10 sock_len = sizeof(cli_addr);
11 while(1) {
12   new_sock = accept(sock_fd,
13         (struct sockaddr *)&cli_addr, &sock_len);
    /* 요청 메시지 처리 */
14    process_request();
15    }
16 }
```

〈그림 2〉 모듈-1 : 웹 서버 초기화

<그림 2>에서 socket()은 TCP/IP 소켓을 생성한 후 bind()는 서버의 IP 주소 및 포트 번호(80)를 할당하고, listen()에서 연결 요청을 기다린다. 이때 웹 브라우저와 동기화시키기 위하여 연결 요청이 들어올 때까지 기다리도록 블록 모드의 소켓을 유지한다. 연결 요청이 들어오면 accept()는 새로운 소켓 식별자를 할당하여 요청 메시지를 처리한다.

2.3.2 모듈-2 : 요청 메시지 검사

HTTP 규약[David et al., 2002]에 의하면, 웹 브라우저로부터 들어오는 요청 메시지(Request) 형식은 <그림 3>과 같이 정의되어 있다.

```
1 Request = Request-Line
2           *((general-header
3           |request-header
4           |entity-header ) CRLF)
5           CRLF
6           [message-body]

7 Request-Line = Method SP Request-URI SP
  HTTP-Version CRLF
```

〈그림 3〉 요청 메시지 형식

<그림 3>에서 요청 메시지는 빈 줄(5번)을 경계로 헤더 부분(1~4번)과 바디 부분(6번)으로 구성 되며 헤더 부분은 Request-Line(1번)과 헤더 정보 부분들(2~4번)로 구성된다. 한편 Request-Line(7번)은 반드시 Method 필드로 시작하여, 공백, Request-URI, 공백, HTTP-Version 순서로 나타난다.

예를 들어, 웹 서버에 연결하기 위한 맨 처음 요청 메시지의 내용은 <그림 4>와 같이 헤더 부분만으로 구성되어 있다.

```
GET /index.htm HTTP/1.0CRLF
Connection: Keep-AliveCRLF
User-Agent: Mozilla/4.5 [en] (Win95; I)CRLF
Pragma: no-cacheCRLF
Host: 10.1.1.11CRLF
Accept: image/gif, image/jpeg, image/pjpeg, */*CRLF
Accept-Encoding: gzipCRLF
Accept-Language: enCRLF
Accept-Charset: iso-8859-1,*,utf-8CRLF
```

〈그림 4〉 웹 문서 요청 메시지 예

<그림 4>에서 헤더 부분의 첫 번째 줄은 GET Method로 index.htm 웹 문서를 요청하는 것으로써 HTTP-Version 1.0을 사용하고 있음을 의미한다. 이어지는 줄들은 웹 서버가 사용할 수 있는 추가적인 정보들이다. 헤더 정보 부분은 헤더 이름, 콜론(:), 공백 그리고 헤더 값으로 구성되고, 각각은 <CR><LF>로 구분된다.

인터넷 가전을 위한 웹 서버에 대한 웹 브라우저로부터의 요청 메시지는 오직 사용자 메뉴를 위한 웹 문서에 대한 요청과 감시 및 제어를 위한 CGI 프로그램 실행 요청뿐이다. 따라서 <그림 5>와 같이 요청 메시지의 첫 번째 줄에 나타나는 파일명의 확장자를 검사함으로써 웹 문서 요청과 CGI 프로그램 실행 요청을 구별하여 각각에 대하여 처리하도록 한다.

```
1 char buffer[512], method[5], uri[256], *file_name;
2 process_request() {/* 요청 메시지 검사 */
3   read(new_sock, buffer, 512);
4   sscanf(buffer, "%s %s", method, uri);
5   file_name = get_filename(); /* 파일명 얻기 */
6   if(strstr(file_name, ".cgi")) /* 파일명 검사 */
7     process_htm(); /* 웹 문서 처리 */
8   else
9     process_cgi(); /* CGI 처리 */
10 }
```

〈그림 5〉 모듈-2 : 요청 메시지 검사

<그림 5>에서 요청 메시지 전체를 buffer에 읽어 들인 다음, buffer로부터 Method와 URI 필드를 분리한다(4번). 이때 URI 필드에는 웹 문서 요청일 경우 파일명만 존재하지만, CGI 요청일 경우에는 파일명에 이어 CGI 프로그램에 전달될 인수들이 존재하기 때문에 URI 필드로부터 파일명만 별도로 분리한다(5번). 또한 URI 필드와 파일명은 전역변수로 선언하여 웹 문서 및 CGI 처리 모듈에서도 참조할 수 있도록 한다.

2.3.3 모듈-3 : 웹 문서 처리

웹 문서 요청에 대한 처리는 간단하다. 요구된 웹 문서 파일의 내용을 <그림 6>과 같이 읽어 HTTP 규약에 정의된 응답 메시지 형식에 맞춰 전송하면 된다.

```
1 int fp;
2 char web_page[1024];
3 struct stat file_info;
4 process_htm() { /* 웹 문서 처리 */
5   fp = open(file_name, O_RDONLY)) /* 웹 문서 열기 */
6   fstat(fp, &file_info);
7   *length = file_info.st_size;
8   read(fp, web_page, *length); /* 웹 문서 읽기 */
9   close(fp);
10  process_response(web_page); /* 응답 메시지 처리 */
11 }
```

〈그림 6〉 모듈-3 : 웹 문서 처리

2.3.4 모듈-4 : CGI 처리

요청 메시지의 URI 필드에 ".cgi"가 포함되어 있으면 CGI 프로그램 실행에 대한 요청으로 취급한다. HTTP 규약에 의하면 "GET", "POST" 등 다양한 종류의 Method가 정의되어 있다. 그러나 "GET"만을 이용해도 사실상 모든 요청이 가능하다[김홍남, 1998]. 특별히 인터넷 가전을 위한 웹 서버에서는 인터넷 게시판과 같은 기능을 요구하지 않기 때문에 리눅스 프로세스의 환경변수

"QUERY_STRING"을 통한 인수전달만으로도 충분하다. 또한 범용이 아닌 특수 목적의 임베디드 웹 서버에서는 일반적으로 CGI 프로그램의 요청을 위한 웹 문서 자체를 웹 서버 프로그래머가 직접 작성하게 되기 때문에 본 논문에서는 "GET" Method만으로 웹 문서 및 CGI 프로그램 실행을 요청하도록 제한하여 <그림 7>과 같이 설계한다.

```
1 process_cgi() { /* CGI 처리 */
2   int pid, pipe_fd[2];
3   char cgi_arg[256];
4   pipe(pipe_fd); /* 파이프 채널 생성 */
5   if((pid =fork()) == 0) {
/* 인수 값 얻기 */
6     char *cgi_arg = get_cgi_arg();
7     char *nargv[] = {file_name, NULL};
/* 환경변수 설정 */
8     char *nenv[] = {"QUERY_STRING="
                    + cgi_arg, NULL};
/* 표준출력을 파이프로 설정 */
9     dup2(pipe_fd[1], stdout);
/* CGI 프로그램 실행 */
10    execve(file_name, nargv, nenv);
11  }
12  else {
/* 자식 프로세스 기다림 */
13    while(wait((int *)0) != pid);
/* 결과 값 읽기 */
14    read(pipe_fd[0], cgi_buf, 256));
/* 응답 메시지 보내기 */
15    process_response(cgi_buf);
16  }
17 }
```

<그림 7> 모듈-4 : CGI 요청 처리

<그림 7>에서 부모/자식 프로세스 간의 통신을 위한 PIPE 채널을 생성한 후(4번), CGI 프로그램 실행을 위한 자식 프로세스를 생성한다(5번). 한편 CGI 요청일 경우, CGI 프로그램에게 전달될 인수 값들이 파일명과 함께 요청 메시지의 ULI 필드에 존재하기 때문에 URI 필드로부터 인수 값을 분리한다(6번). 자식 프로세스는 실행해야할 CGI 프로그램 명과 환경변수(QUERY_STRING) 값을 설정한다(7번). 또한 실행 결과 값을 부모 프로세스에게 쉽게 전달할 수 있도록 표준출력을 파이프 채널로 설정한 후(8번), execve()을 이용하여 CGI 프로그램을 실행한다(9번). 한편, 부모 프로세스는 자식 프로세스의 실행을 기다린 후, 자식 프로세스의 실행 결과 값을 파이프 채널로부터 읽어온다.

결국 웹 서버는 환경 변수(QUERY_STRING)를 통하여 CGI 프로세스에게 인수 값을 전달하고, CGI 프로세스의 실행 결과는 파이프 채널을 통하여 웹 서버 프로세스에게 전달된다. 여기에서 PIPE 채널을 통한 IPC 기법을 사용하는 이유는 기존의 범용 CGI 프로그램들이 PIPE 채널을 사용하고 있기 때문에 이들과의 호환성을 유지함으로써 확장성을 제공하기 위해서이다.

2.3.5 모듈-5 : 응답 메시지 보내기

HTTP 규약에 의하면, 웹 브라우저로 보내는 응답 메시지(Response) 형식은 <그림 8>과 같이 정의되어 있다.

```
1 Response = Status-Line
2            *((general-header
3            |response-header
4            |entity-header ) CRLF)
5            CRLF
6            [message-body]

7 Status-Line = HTTP-Version SP
  Status-Code SP Reason-Phrase CRLF
```

<그림 8> 응답 메시지 형식

<그림 8>에서 응답 메시지는 빈 줄(5번)을 경계로 헤더 부분(1~4번)과 바디 부분(6번)으로 구성 되어 있다. 한편, 헤더 부분의 Status-Line(7번)에는 HTTP-Version 필드로 시작하여, 공백, Status-Code, 공백, Reason-Phrase로 구성된다. HTTP 규약에 의하면, 요청 메시지에 대한 처리

상태를 표시하는 다양한 종류의 Status-Code 및 Reason-Phrase가 정의되어 있다. 그러나 이러한 정보는 오류에 대한 사용자의 편의성을 위한 것으로 모두 다 지원할 필요는 없다. 따라서 본 논문에서는 범용이 아닌 특수 목적의 임베디드 웹 서버로써 반드시 정상적으로 처리된다는 전제하에서 Status-Code "200"에 Reason-Phrase "OK" 만으로 제한하여 <그림 9>와 같이 헤더 부분을 전송한 후 바디 부분을 전송하도록 설계한다.

```
1 response_request(char *body) {
2   char *header = "HTTP/1.0 200 OK₩n"
                   "Content-type: text/html₩n"
                   "₩n";
/* 헤더 부분 전송 */
3   write(new_sock, header, strlen(header));
/*바디 부분 정송 */
4   write(new_sock, body, strlrn(body));
5 }
```

<그림 9> 모듈-5 : 응답 메시지 처리

3. 구현 및 비교평가

인터넷 가전의 원격 감시 및 제어를 위해 제시한 웹 서버의 적용 가능성을 검토하기 위하여 각각의 모듈을 구현하고 다음과 같은 환경에서 기본적인 기능들을 실험하였다.

<표 1> 임베디드 보드(EZ-Board) 사양

항 목	사 양
CPU	StrongARM(SA1110)
RAM	32M SDRAM
ROM	16M Flash
Network	CS8900 10-Mbps
USB	USB Client
Serial	RS-232C(3 port)
Kernel	Linux-2.4.18
File System	ramdisk
Boot Loader	ezboot

3.1 구현 환경

제시한 임베디드 웹 서버의 모듈들은 리눅스 커널(2.4.18)에서 C언어로 구현하고, (주)J.D&T에서 제조한 시험용 임베디드 보드[10]에 설치하였다. 시험용 임베디드 보드의 하드웨어 및 소프트웨어 사양은 <표 1>과 같다.

3.2 실험 방법

3.2.1 메뉴 제공을 위한 웹 문서 요청

간단한 형태의 웹 문서(index.htm)를 작성한 후, MS Explorer에서 접속한다. 이 요청이 처리되는 과정은 다음과 같다.

(1) 요청 메시지에 대한 검사 과정

<그림 5>에서 읽어 들인 요청 메시지의 내용(buffer)을 확인한다. 메시지 내용 중 파일명의 확장자가 ".cgi"가 아님으로 웹 문서 처리함수 "process_htm()"를 호출한다.

(2) 웹 문서 처리 과정

<그림 6>에서 "index.htm" 문서를 열고 "web_page"에 읽어 들인 후, 응답 메시지 처리함수 "process_response(web_page)"를 호출한다.

(3) 응답 메시지 처리 과정

<그림 9>에서 정의된 응답 메시지의 헤더 부분이 전송된 후, 곧바로 "web_page"의 내용을 전송한다.

3.2.2 감시 및 제어를 위한 CGI 프로그램 실행 요청

환경변수 "QUERY_STRING"으로 전달된 두 수의 합을 구하는 CGI 프로그램(sum.cgi)을 작성한다. 임의의 두 수를 입력받아 GET 방식으로 "sum.cgi" 프로그램 실행을 요청하는 웹 문서(sum.htm)를 작성하여 MS Explorer에서 접속한 후, 임의의 두 수를 입력하고 "sum.cgi" 프로그램 실행 결과 값을 얻기 위한 "요청버튼"을 클릭 한

다. 이 요청이 처리되는 과정은 다음과 같다.

(1) 요청 메시지에 대한 검사 과정

<그림 5>에서 읽어 들인 요청 메시지의 내용(buffer)을 확인한다. 메시지 내용 중 파일명의 확장자가 ".cgi"이므로 CGI 처리함수 "process_cgi()"를 호출한다.

(2) CGI 프로그램 실행 과정

<그림 7>에서 새로운 프로세스를 생성한 후, execve()함수를 이용하여 "sum.cgi"를 실행시킨다. 이때 환경변수 "QUERY_STRING"에 전달된 두 수를 부여하고, CGI 프로그램의 실행 결과 값은 파이프 채널을 통하여 서버 프로세스에게 전달된다. 서버 프로세스는 파이프 채널로부터 결과값을 "cgi_buf"에 읽어 들인 후, 응답 메시지 처리 함수 "process_response (cgi_buf)"를 호출한다.

(3) 응답 메시지 처리 과정

<그림 9>에서 정의된 응답 메시지의 헤더 부분이 전송된 후, 곧바로 "cgi_buf"의 내용을 전송한다.

3.2.3 사용자 인증 처리를 위한 기존 CGI 프로그램 이용

기존 CGI 프로그램과 호환성을 검토하기 위하여 Rob McCool이 개발한 "htpasswd.c"과 "htpasswd.htm" 프로그램[김홍남, 1998]을 이용하여 사용자 인증처리 기능을 추가해 본다. "htpasswd.c" 프로그램은 환경변수 "QUERY_STRING"으로 전달된 사용자 ID와 비밀번호를 관리한다. 한편 "htpasswd.htm" 프로그램은 사용자 관리를 위한 웹 문서이다.

사용자 ID와 비밀번호를 입력받아 GET 방식으로 "htpasswd.cgi" 프로그램 실행을 요청하는 웹 문서(htpasswd.htm)를 작성하여 MS Explorer에서 접속한 후, 사용자 ID와 비밀번호를 입력하고 "확인버튼"을 클릭 한다. 이 요청에 대한 처리과정은 3.2.2에서 두 수 대신 사용자 ID와 비밀번호를 사용하는 것 외에는 동일하다.

3.3 비교 평가

제시한 임베디드 웹 서버의 설계 목표인 소형화, 이식성 및 확장성을 평가하기 위하여 기존의 Boa 웹 서버와 기능 및 구조적인 요소들을 비교해 보았다.

<표 2>에서 알 수 있듯이 기존의 웹 서버(Boa-0.94.13)는 속도, 보안 등의 설계목표와 범용을 위한 모든 HTTP 규약 지원으로 이미지가 크고, 다중 사용자를 지원하기 위한 Single-Tasking with Multiplexing I/O 구조로 너무 복잡하다. 반면 본 논문에서 제시한 웹 서버의 이미지 크기는 14,098B로써 기존의 Boa 웹 서버

<표 2> 임베디드 웹 서버의 비교

항 목	기존(Boa-0.94.13)	제시한 웹 서버
설계 목표	속도, 보안, 강인성, 이식성	소형, 이식성, 확장성
이미지 크기	251,278B	14,098B
I/O 구조 (사용자)	Single-Tasking w/ Multiplexing (다중 사용자)	Single-Tasking w/o Multiplexing (단일 사용자)
Method	GET, HEAD, POST, PUT, etc.	GET (only)
Media Type	MIME Type	text/html, plain
Status Code	Informational 1xx, Successful 2xx, Redirectional 3xx, Client Error 4xx, Server Error 5xx	Successful 200 OK (only)
CGI 프로그램 실행	fork()/execve() pipe()	fork()/execve() pipe()
파일명 파싱	yacc 이용	strstr()

(251,278B)에 비해 매우 작으며, 메뉴 제공을 위한 웹 페이지 요청 및 감시 및 제어를 위한 CGI 요청 처리는 물론 사용자 인증과 같은 기존의 CGI 프로그램을 쉽게 이용할 수 있음을 확인하였다.

4. 결 론

대부분의 기존 웹 서버들은 범용성, 속도, 보안 및 성능 등을 고려하여 설계되었기 때문에 그 규모가 커지고 구조가 복잡하여 제한된 기능만을 요구하는 임베디드 시스템에 적용하기에는 오버헤드가 있다. 이러한 문제점을 해결하기 위하여 본 논문에서는 인터넷에 연결된 가전 제어를 위한 임베디드 시스템에 적합한 웹 서버를 제시하였다. 또한 제시한 웹 서버의 적용 가능성을 검토하기 위하여 리눅스 커널(2.4.18)에서 각각의 모듈들을 구현하여 MS Explorer 6.0으로 실험하였다.

기본적으로 인터넷 가전의 감시 및 제어를 위한 웹 서버는 사용자 메뉴를 위한 웹 문서 참조와 감시 및 제어를 위한 CGI 프로그램 실행 요청에 대한 서비스 제공으로 그 기능을 단순화함으로써 크기를 최소화하고, 모듈별로 설계함으로써 비교적 단순 기능만을 요구하는 임베디드 웹 서버에 매우 적합할 것으로 본다.

참 고 문 헌

[1] 강선례, 신동하, "내장형 시스템용 웹 서버 개발", *정보처리학회추계발표 논문집*, 제10권 제2호, 2003. pp. 401-404.

[2] 김두현, 김정국, "임베디드 소프트웨어 플랫폼 표준화 동향", *정보처리학회지*, 제11권 제6호, 2004, pp. 33-41.

[3] 이봉규, 송지영, "디지털 홈 구현을 위한 홈 서버 및 정보가전 단말 기술 동향", *정보처리학회지*, 제11권 제3호, 2004, pp. 46-54.

[4] 임채덕, "임베디드 소프트웨어 개발 도구", *정보처리학회지*, 제11권 제6호, 2004, pp. 76-85.

[5] 김홍남, *CGI 파워 프로그래밍*, 대림출판사, 1998.

[6] 박종진, 이동은, 이형수, *임베디드 웹 서버를 위한 TCP/IP*, 에이콘 CMP Books, 2002.

[7] David Gourley & Brian Totty, *HTTP: The Definitive Guide*, O'Reilly, 2002.

[8] John Lombardo, *Embedded Linux*, New Riders, 2001.

[9] Hypertext Transfer Protocol--HTTP/1.1, http://www.w3.org/Protocols.

[10] Falinux, http://www.falinux.com.

[11] The source code of Boa Web Server, http://www.boa.org.

[12] The source code of Goahead, http://www.goahead.org.

저자소개

임 성 락

저자는 1979년 서강대학교에서 전자공학 학사를 취득했고, 1983년 서울대학교에서 컴퓨터공학 석사를 취득하였으며, 1992년 서울대학교에서 컴퓨터공학 박사를 취득하였다. 1983년부터 1990년까지 금성반도체 선임연구원으로 재직하였으며 1993년부터 현재까지 호서대학교 컴퓨터공학과 교수로 재직하고 있다.

◈ 이 논문은 2005년 7월 5일 접수하여 3차 수정을 거쳐 2005년 12월 5일 게재확정되었습니다.