

## 2018년도 2학기 오픈소스프로젝트 (1분반)

### 가벼운 IoT 시스템 구현

#### 1. 개요

- POSIX 표준이 제공되는 환경에서 C언어를 사용하여 간단한 IoT 시스템을 구현한다. 이 시스템은 센서 (sensor) 정보를 서버에 보내는 센서 프로그램, 센서 정보를 저장하는 서버 프로그램, 그리고 저장된 정보를 열람하는 모니터 프로그램으로 구성된다.
- 기존의 통신 체계를 그대로 활용할 수 있는 웹 서버/클라이언트들 간의 통신 (HTTP)으로 세 종류의 프로그램들 간의 통신이 이루어진다.
- 웹 서버는 여러 클라이언트들로부터 오는 데이터를 효과적으로 수신하기 위해 multi-thread 기능으로 구현한다.
- 본 과목과 관련된 오픈소스 또는 표준 프로토콜
  - HTTP protocol
  - Robust I/O
  - mySQL (또는 noSQL 계열)

#### 2. Web server와 HTTP protocol에 대한 간단한 소개

- Web server와 web client간의 통신은 다음과 같다 (그림 1). 요청과 응답은 HTTP (Hypertext Transfer Protocol)라는 텍스트 기반의 프로토콜을 사용한다.
  1. Web client는 web server에게 인터넷 연결을 요청한다.
  2. Web client는 HTTP를 통해 web page를 요청한다.
  3. Web server는 요청된 내용을 회신하고 연결을 닫는다.
  4. Web client는 그 내용을 읽고 화면에 표시한다.

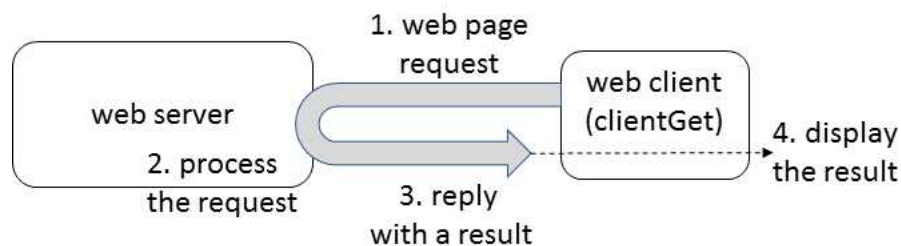


그림 1 web communication

- Web server의 요청 처리 과정
  - Web server의 정보단위는 파일이며 웹 페이지나, 그림은 각각 하나의 파일과 연관되어 있다.
  - Client가 server의 디스크에 있는 특정 파일을 요청하면 이는 파일이라는 '정적 요청'이라고 판단한다.
  - Client가 server의 디스크에 저장된 어떤 실행 파일을 수행하고 그 결과를 요청하면 이는 '동적 요청'이라고 판단한다.

- 요청에 있는 URL (Universal Resource Locator)이라는 필드는 정적 요청의 파일이나 동적 요청의 실행 파일 이름을 담고 있다.
- URL 예제1 : kit.kumoh.ac.kr:80/~choety/index.html
  - 이는 “kit.kumoh.ac.kr”이란 인터넷 호스트에 있는 “choety”란 사용자의 디렉터리에 있는 “/index.html”이란 HTML 파일을 ‘정적 요청’한다는 의미이며, 이 호스트의 web server는 80번 port로 부터 client의 요청을 기다린다. Port는 다른 컴퓨터에 있는 프로세스들 간의 통신을 위한 주소이며 ftp는 21번, ssh는 22번, telnet은 23번, 그리고 http는 80번을 사용한다.
- URL 예제2 : kit.kumoh.ac.kr:80/~choety/output.cgi?name=hobby&value=100
  - 이는 해당 인터넷 호스트에 있는 output.cgi라는 실행 파일을 서버가 수행하고 그 결과를 클라이언트가 요구하는 ‘동적 요청’이며 그 파일명 뒤에 여러 개의 argument들을 포함시킬 수 있다.
  - ‘?’ 문자는 파일명과 argument를 구분한다.
  - ‘&’ 문자는 argument들을 서로 구분한다.
  - Argument들은 string의 형태로 QUERY\_STRING 환경변수의 일부분으로 CGI 프로그램에게 전달된다.
  - 정적 요청의 경우 header에 필요한 정보를 넣고, body에 요청한 파일의 내용을 넣어 client에게 전달한다.
  - 동적 요청의 경우 server는 argument들을 요청된 실행 파일에 전달한 후 그 실행 파일을 수행한다. Server는 client에게 header를 보낸 후 실행 파일의 출력 결과를 client에게 forward한다. 동적 요청에 사용되는 실행 파일들은 대부분 표준 출력(stdout)으로 결과를 출력한다. Server는 이 실행 파일의 파일 기술자 (file descriptor)를 client로 향하는 socket 통신의 파일 기술자로 변경시켜 실행 파일의 표준 출력 결과를 client가 받도록 한다.
- HTTP 요청 (request) 형식
  - Web browser (client)에서 web server에게 전달하는 메시지의 형식
  - 그림 2와 같이 1개의 request line, 0개 이상의 request header, 1개의 empty text line, 0개 또는 1개의 request body로 구성된다.

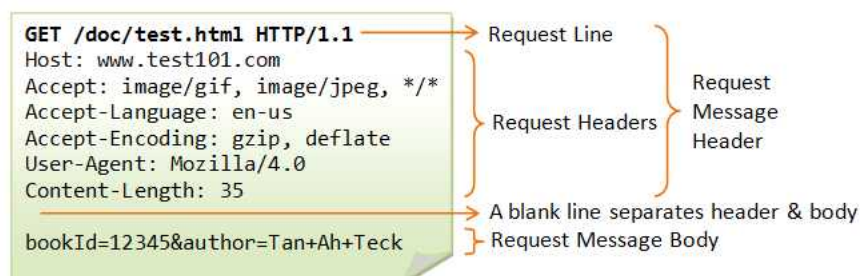


그림 2 HTTP request format

- Request line : [method] [uri] [version]
  - [method] : (주로) GET, (가끔) POST, OPTIONS, PUT

- [uri] : 파일명과 argument들
  - [version] : web client가 사용하는 HTTP 프로토콜의 버전 (HTTP/1.0 또는 HTTP/1.1)
  - Request header : 여러 개의 option들로 구성. 아래는 몇 개의 예시이다. 모든 option은 [https://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_header\\_fields](https://en.wikipedia.org/wiki/List_of_HTTP_header_fields) 를 참고하시오.
    - Host: kit.kumoh.ac.kr:80
    - Content-Length: 451
    - Accept-Charset: utf-8
  - Empty text line : “\r\n”
  - Request body : POST method의 경우에 Content-Length 길이만큼의 data가 request body에 존재한다. 따라서 POST method가 처리될 때나 HTTP 응답이 보내질 때는 Content-Length가 중요한 역할을 한다.
- HTTP 응답 (response) 형식
    - HTTP 요청의 결과로 web server에서 web client에게 전달하는 메시지의 형식
    - 1개의 response line, 0개 이상의 response header, 1개의 empty text line, 1개의 response body로 구성된다.
      - Response line : [version] [status] [message]
        - [status] : 3자리 양의 정수 (POST method의 요청에 문제가 없는 경우 200, Forbidden인 경우 403, Not found인 경우 404, 등)
      - Response header : 아래와 같은 option들로 구성된다.
        - Content-Type : client에게 response body에 있는 데이터가 어떤 MIME 형식인지 (html, gif 등) 알린다.
        - Content-Length : response body가 몇 byte 길이인지 알린다.
    - 그림 3은 성공적인 HTTP 응답의 한 예시이다.

```

HTTP/1.1 200 OK
Content-Length: 58
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>

```

그림 3 HTTP 응답 예시

- Telnet을 통해 HTTP 프로토콜 내용을 직접 보내는 방법

- 그림 4와 같이 일반 shell에서 다음을 타이핑하면 상대 웹 서버는 HTTP 프로토콜로 인식하고 <http://kit.kumoh.ac.kr/index.html> 파일의 내용을 반환한다.

```
$ telnet kit.kumoh.ac.kr 80
.....
GET / HTTP/1.1
host: pintos.kumoh.ac.kr
HTTP/1.1 200 OK
Date: Wed, 07 Sep 2016 16:04:22 GMT
Server: Apache/2.2.11 (Unix) PHP/5.2.6 mod_jk/1.2.28
X-Powered-By: PHP/5.2.6
Content-Length: 96
Content-Type: text/html

<html>
<head>
<meta http-equiv=refresh content="0; url=http://www.kumoh.ac.kr">
</head>
</html>
Connection closed by foreign host.
```

그림 4 HTTP 프로토콜 내용을 telnet을 통해 보기

- 웹주소와 HTTP 프로토콜의 관계
  - Web browser가 웹주소 <http://kit.kumoh.ac.kr/~choety/> 을 사용자로부터 받으면 그림 5와 같은 내용을 web server인 kit.kumoh.ac.kr에 보낸다. 일반적인 GET method처럼 request body 부분은 비어있다.

```
GET /~choety/ HTTP/1.1\r\n
Host: kit.kumoh.ac.kr\r\n
Connection: keep-alive\r\n
....
\r\n
```

그림 5 <http://kit.kumoh.ac.kr/~choety/> 의 HTTP protocol

- 웹 주소 <http://kit.kumoh.ac.kr/output.cgi?data=test&value=100> 을 web browser 는 그림 6와 같은 HTTP protocol로 바꾸어 보낸다. 이는 동적 요청으로 web server 에 있는 output.cgi 파일을 수행시키며 argument로 data=test&value=100를 전달한다. 직전 예제와 마찬가지로 request body 부분은 비어있다.

```
GET /output.cgi?data=test&value=100 HTTP/1.1\r\n
Host: kit.kumoh.ac.kr\r\n
Connection: keep-alive\r\n
....
\r\n
```

그림 6 <http://kit.kumoh.ac.kr/output.cgi> 의 HTTP protocol

- Request body가 있는 웹 요청
  - 그림 7와 같은 코드로 구성된 웹 페이지는 그림 8와 같은 모양을 가지며 ‘Submit’ 버튼을 클릭하면 그림 9와 같은 내용이 해당 서버에 전달된다. Form 태그에서 POST method로 전송하겠다는 의미로 작성되었으며, fname이라는 변수명과 해당 값 그리고 lname이라는 변수명과 해당 값이 submit 버튼을 눌렀을 때 전송된다. 전송되는 값들은 POST method의 경우 request body에 넣어진다. 또한 request body의 길이는 request header의 Content-Length 옵션에 주어진다.

```
<form action="/dataPost.cgi" method="post">
  Time: <input type="text" name="fname"><br>
  Value: <input type="text" name="lname"><br>
  <input type="submit" value="Submit">
</form>
```

그림 7 form tag로 이루어진 http code

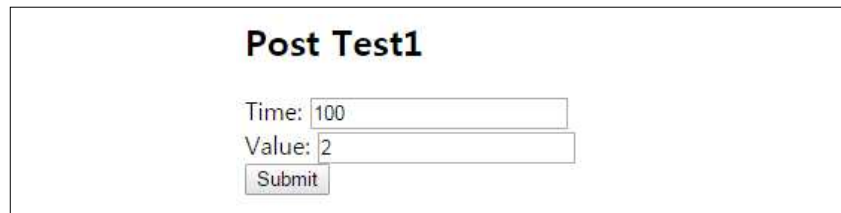


그림 8 그림 3의 코드가 web browser에서 표현된 형태

```
POST /dataPost.cgi HTTP/1.1\r\n
Host: 127.0.0.1:8080\r\n
Connection: keep-alive\r\n
Content-Length: 17\r\n
...
\r\n
fname=100&lname=2
```

그림 9 그림 3의 코드 수행으로 인한 HTTP protocol

### 3. 기본적으로 제공되는 프로그램

- 제공되는 web server 및 client 사용 과정
  - 자신의 리눅스 머신에 make와 C 언어 컴파일러가 가능하도록 시스템이 설정되어 있어야 한다.
    - 설치: `sudo apt-get install gcc make`
  - LMS에서 소스파일을 다운받아 자신의 리눅스 작업용 디렉터리에 저장한다.
    - 설치: `sudo apt-get install unzip`
    - 수행: `unzip osp.zip -d Project`
  - **make** 명령을 수행하면 컴파일이 완료된다.
    - `cd Project`
    - `make`
  - **make clean** 명령을 수행하면 목적파일인 .o 파일들이 제거된다. 이 명령은 다른 운영 체제 환경이나 처음부터 컴파일 할 때 필요하다.
  - 이 web server를 수행할 때는 client로부터 요청을 기다리는 port 번호가 필요하다. config-ws.txt 텍스트 파일에 port 번호가 있으며 이를 바꿈으로서 요청을 기다리는 port 번호를 변경할 수 있다.
    - 가급적이면 기존 port 번호와의 충돌을 피하기 위해 2000이상의 값을 사용하는 것을 권장한다.
    - Web browser에서 이 server에 접속할 때는 같은 port 번호를 사용하여 접속해야 한다.
    - 제공되는 config-ws.txt에는 8080 port number가 있다.
  - Web browser에게 최소 하나의 web page는 제공해야 하므로 소스코드와 같이 제공되는 index.html과 test.html 파일이 실행파일 server와 같이 있는지 확인한다.
  - 자신의 리눅스 머신에서 basic web server를 다음과 같이 수행시킬 수 있다.  
`./server`
  - 간편하게 테스트를 하기 위해 동일한 머신에서 web browser로 index.html 파일을 보기 위해 다음의 URL을 사용하면 그림 10과 같은 결과를 볼 수 있다.

<http://127.0.0.1:8080/>



그림 10 제공되는 web server를 일반 web browser에서 보기

- 이제 basic web client를 실행시켜 보자. Basic web server가 같은 컴퓨터 (127.0.0.1 IP 주소를 가진 컴퓨터)에서 수행되고 있고, config-cg.txt 파일에 (127.0.0.1\n 8080\n /index.html)이 저장되어 있다고 하자. 이 경우 다음과 같이 basic web client를 실행시킬 수 있다. clientGet 파일은 GET method를 요청으로 1번 보내고 그 응답을 받아 출력한 뒤 종료하는 역할을 한다.

```
./clientGet
```

- 두 번째로 제공되는 web client인 clientPost는 POST method를 요청으로 1번 보내는 용도로 만들어졌다. 이 파일은 config-cp.txt 파일이 필요하며 다음과 같이 실행할 수 있다. 현재 동적 요청이 구현되지 않은 상태이므로 “HTTP/1.0 404 Not found” 에러가 발생한다.

```
./clientPost
```

- 프로그램의 소스코드

- 약 1500줄의 C 코드와 헤더 파일에 몇 개의 helper 함수들로 구현되어 있다.
  - 제공되는 프로그램은 다양한 경우를 고려하지 않는다. (예를 들어, web client가 connection을 close하는 명령어를 보내면 basic web server는 다운될 수 있다. 하지만 이것을 고치는 것이 이 과제의 목표는 아니다.)
- 제공되는 basic web server의 소스코드인 server.c는 코드 이해를 돕기 위해 최소한의 기능만 포함되어 있다. 다음과 같은 과정으로 수행된다.
  - config-ws.txt 파일로부터 자신이 수신할 port 번호를 읽는다.
  - port를 통해 요청을 기다린다.
  - 요청이 도착하면 requestHandle()을 불러 요청을 처리하고, 처리가 끝나면 다시 port를 통해 요청을 기다린다.
- 소스코드 request.c에는 requestHandle()내에서 다음과 같은 작업들이 수행된다.
  - Client로부터 도착한 요청이 정적 요청 인지 동적 요청인지 구분하여 해당 함수가 처리하도록 한다.
    - requestServeStatic(): 정적 요청 처리 함수. 과제에서는 건드릴 필요가 없다.
    - requestServeDynamic(): 동적 요청 처리 함수. 과제의 핵심 부분이다. 현재 dummy 응답만 보내는 코드로 되어 있다. 과제 진행 과정에서 이 코드 대신 요청된 파일을 수행하는 코드가 채워져야 한다.
    - requestType(): 프로토콜의 첫 번째 줄을 읽고, 정적 요청과 동적 요청을 구분하는 함수인 parseURI()를 호출한다.
    - parseURI(): 정적/동적 요청을 구분한다. 과제 진행 과정에서 argument가 있는 경우 filename으로부터 분리하는 작업을 수행하도록 개선되어야 한다.(예: /output.cgi?data=100 → (/output.cgi, data=100))
    - Client로부터의 요청이 에러를 발생시키면 에러 코드를 requestError() 함수를 통해 client에 리턴한다.
  - 현재 코드는 HTTP method들 중 GET과 POST method만 인식한다. (requestHandle() 함수)
  - 적은 개수의 Content-Type만 인식한다. (requestReadhdrs() 함수)
    - 해당 함수는 현재 Content-Length: option만 인식한다.

- 소스코드 stems.c를 통해 다음과 같은 함수들이 제공된다.
  - helper 함수 (system call에 대한 wrapper)
    - 요청한 system call이 error code를 리턴 했는지 확인하고 그럴 경우 즉시 종료 하기 위해 사용한다. (예: Fork(), Open(), Dup2())
    - System call을 사용할 때는 그 결과의 에러 유무에 따라 에러 처리를 위한 코드가 필요하지만 이 코드들이 소스 코드 곳곳에 놓이면 소스 코드의 가독성을 떨어트리는 문제가 있다. 이를 해결하기 위해 wrapper 함수를 정의하고 그 내부에 system call을 부른 뒤 그 결과값에 따라 unix\_error()나 posix\_error() 함수를 호출한다.
    - 보통 wrapper 함수의 이름은 해당 system call 이름에서 첫 문자를 대문자로 바꾸어 사용한다. (예: socket() → Socket(), bind() → Bind(), listen() → Listen())
    - 추가로 system call을 사용해야 할 경우 wrapper 함수를 만들어 사용할 것을 권장한다.
  - RIO 패키지
    - Robust I/O 함수들은 socket 통신에서 상대방이 보내는 내용을 buffer에 저장해 두고 필요한 크기만큼 반환함으로써 성능을 향상시키는 버퍼링 기능, n byte만큼 읽는 기능, 그리고 '\n'을 만날 때까지 1줄 단위로 읽는 기능을 제공한다. 위 세 기능을 제공하지 않는 함수들도 있기 때문에 이를 구분하기 위해 버퍼링은 'b', 크기 만큼 읽는 기능은 'n', 그리고 줄 단위 읽는 기능은 'line'이라는 단어가 read() 나 write() 함수명에 추가된다. 또한 각 robust I/O 함수들에 대한 wrapper 함수가 첫 글자가 대문자의 형태로 제공된다.
    - 구현된 RIO 함수: rio\_readn(), rio\_writen(), rio\_readinitb(), rio\_readnb(), rio\_readlineb()
    - RIO wrapper 함수: Rio\_readn(), Rio\_writen(), Rio\_readinitb(), Rio\_readnb(), Rio\_readlineb()
    - Rio\_readn(fd, buf, n): file descriptor fd로부터 n byte만큼을 읽는다. 시그널 인터럽트로 중단되었을 때 다시 시작한다는 점을 제외하고는 read()와 동일하다.
    - Rio\_writen(fd, buf, n): file descriptor fd로 n byte만큼을 쓴다. 시그널 인터럽트로 중단되었을 때 다시 시작한다는 점을 제외하고는 write()와 동일하다.
    - Rio\_readinitb(rp): Rio 구조체 \*rp를 초기화한다.
    - Rio\_readnb(rp, buf, n): Rio 구조체 \*rp내의 buffer를 통해 n byte만큼 읽는다. 반환값이 n보다 작으면 해당 file descriptor로부터 더 이상 읽을게 없다는 의미이다.
    - Rio\_readrestb(rp, buf): Rio 구조체 \*rp내의 buffer에 남은 내용을 읽는다.
    - Rio\_readlineb(rp, buf, n): Rio 구조체 \*rp내의 buffer를 통해 n byte만큼 읽는데 중간에 '\n'이 나오면 거기까지만 읽는다.
- 소스코드 clientGet.c는 GET method로 된 요청을 생성하는 간단한 web client 프로그램이다.
  - config-cg.txt 파일로부터 표 1과 같은 내용을 읽는다.



형식	설명	예시
server IP addr.	web server의 IP 주소	127.0.0.1
server port no.	web server의 port 번호	8080
CGI program	수행될 dataPost.cgi의 경로와 이름	/dataGet.cgi

표 1 config-cg.txt의 내용

- 현재 web server는 동적 요청은 수행할 수 없으므로 clientGet을 수행하면 응답은 받지만 “Current version does not support CGI program.”라는 문장이 담기게 된다.
  - 하지만, web server는 존재하는 html 파일에 대한 정적 요청을 수행할 수 있으므로 “/dataGet.cgi”를 정적 요청 URI (예: / 또는 /test.html)로 변경하면 해당 파일을 전달한다. clientGet은 web server로부터 받은 결과를 화면에 출력한다.
- 과제 진행 과정에서 사용자의 명령어 처리 shell을 구현하고 이를 통해 요청을 보내고 그 결과를 화면에 출력하는 기능이 추가되어야 한다.
- 소스코드 clientPost.c는 POST method로 된 요청을 보내는 web client이다.
  - config-cp.txt 파일에서 표 2의 내용을 읽는다. 여기서 time 값은 1970년 1월 1일부터 경과한 시간을 초단위로 표현한 것이다. C 언어에서는 time() 함수를 사용해서 얻을 수 있다.

형식	설명	예시
sensor name	sensor 호칭	dummy
server IP addr.	web server의 IP 주소	127.0.0.1
server port no.	web server의 port 번호	8080
CGI program	수행될 dataPost.cgi의 경로	/dataPost.cgi
time	현재시각	1535713769
value	센서에서 읽은 값	22.4

표 2 config-cp.txt의 내용

- argument를 POST method를 사용하여 web server에게 보낸다. 표 2의 예시대로 읽었다면 “name=temperature&time=1535713769&value=22.4”를 보낸다.
- web server가 아직 동적 요청을 처리하지 못하기 때문에 dummy로 된 응답만을 받아 화면에 출력한다.
- 과제 진행 과정에서 난수를 생성하여 POST의 argument로 보내는 코드를 생성해야 한다.
- 소스코드 dataGet.c는 clientGet.c의 동적요청에 따라 server가 수행시킬 CGI 프로그램이다.
  - 현재는 입력받은 argument를 html의 형태로 return하는 코드만 포함되어 있다.
  - 과제가 진행됨에 따라 DB로부터 요청된 부분을 읽어 return하는 코드를 구현해야 한다.
- 소스코드 dataPost.c는 clientPost.c의 동적요청에 따라 server 측에서 수행된다.
- 프로그램 구조
  - 그림 11은 기본적으로 제공되는 프로그램의 구조를 나타낸다. 각 프로세스들은 다른 컴

퓨터에서 실행가능하며 각 프로세스에 해당하는 config 파일이 같은 디렉터리에 있어야 한다. 현재 server는 GET method를 통한 정적 요청만 처리되며 동적 요청의 경우 200번 응답과 정해진 에러 메시지만 리턴된다.

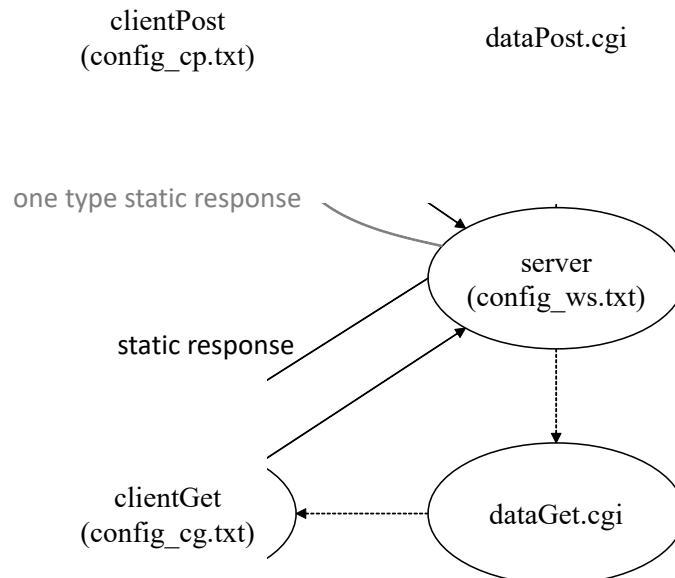


그림 11 제공되는 프로그램 구조

- 그림 11에서 server와 dataGet.cgi 사이에 있는 점선 화살표는 아직 server에서 dataGet.cgi를 호출하는 부분을 구현하지 않았음을 나타낸다. 또한 dataGet.cgi에서 clientGet으로 향하는 화살표도 구현되지 않았기 때문에 점선으로 표시되어 있다. 이는 dataPost.cgi의 경우에도 마찬가지이다. clientPost는 server로 POST method 요청은 보내지만 server가 동적 요청은 처리하지 못하므로 간단한 정적 응답만 보내준다. 해당 그림의 one type static response이 이 정적 응답을 의미한다.

#### 4. 본 과제에서 추가할 내용

- 개요
  - Web client와 server 양쪽 코드에 기능을 확장
    - Web client (clientGet)에는 commend line interface로 사용자의 명령어를 입력받고 서버에 요청하는 기능 추가
    - Web client (clientPost)에는 random sensor data를 주기적으로 생성하여 서버에 전달하는 기능을 추가
      - 이후에 라즈베리 파이에 온습도 센서를 설치하고 GPIO를 통해 그 값을 읽어 서버에 전달하는 기능을 추가
    - Web server에는 동적 요청을 처리하여 DB에 넣거나 읽어서 반환하는 기능 추가
    - Alarm system을 추가하여 web server에서 threshold 보다 큰 값이 수신되었을 때 clientGet측에 전달하여 화면에 출력
  - 그림 12는 본 과제의 최종 형태를 보여준다.

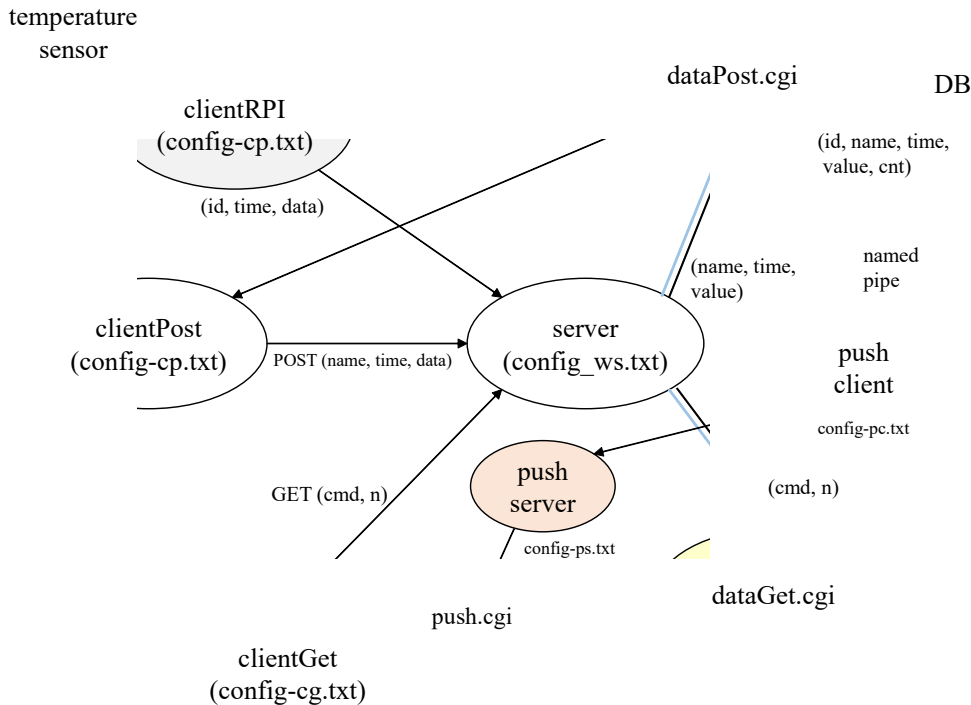


그림 12 본 과제의 최종 형태

#### 4.1. Web server가 동적 요청 수행을 위한 CGI 프로그램 수행

- Web server가 동적요청에 해당하는 프로세스를 수행하도록 하시오.
  - 제공되는 server 코드는 clientGet의 정적 요청 (index.html)은 제대로 처리하지만 clientGet과 clientPost의 동적 요청에 대해서는 에러 메시지만을 보낸다. 본 단계에서 해야 할 일은 표 2에 해당하는 설정 파일로 clientPost가 수행된다고 할 때 dataPost.cgi가 실행되도록 하는 것이다. 마찬가지로 표 1에 해당하는 설정 파일로 clientGet이 수행되면 server가 dataGet.cgi를 실행하도록 한다. server는 CGI 프로그램을 수행시키므로 더 이상 에러 메시지가 포함된 응답을 보내지 않는다.
  - 본 단계가 제대로 수행되더라도 parameter들이 CGI 프로그램에 전달되지 못하므로 동적 요청을 완벽하게 처리하지는 못한다. 해당 CGI 프로그램은 의미가 있는 parameter를 받지 못한 상태로 메시지를 자신이 수행되는 터미널에 출력하므로, server가 이 CGI 프로그램의 출력을 client로 향하는 socket descriptor로 바꾸도록 한다.
  - 파일 기술자 변경을 통한 CGI 실행 결과 반환
    - Web client와 server 사이에 이동되는 정보의 경로는 client에서 발생해서 server를 거쳐 CGI 프로그램을 통해 client에게 반환됨으로써 완성된다. CGI 프로그램의 결과는 일반적으로 stdout descriptor로 출력된다. 따라서 stdout descriptor로 출력되는 방향을 client와 연결한 socket descriptor로 변경해야 한다 (그림 13 참조). Client와 연결된 socket은 request.c 파일의 requestHandle() 함수에 있는 첫 번째 parameter인 connfd를 의미한다. Accept() 함수가 완료되면 web client로부터의 요청이 연결된 증거로서 둘 간의 통신을 위한 새로운 socket 또는 file descriptor가 할당되는데 connfd가 이 새로운 socket descriptor이다. 본 과제에서는 Dup2() 시스템

콜을 이용하여 출력의 방향을 변경하도록 한다. stdout (file descriptor로는 STDOUT\_FILENO)과 socket은 모두 file descriptor의 일종이므로 Dup2() 시스템콜로 제어할 수 있다. CGI 프로그램의 결과를 파일에 저장한 뒤 보내는 방식을 사용하면 감점 처리한다.

- 아직 CGI 프로그램은 구체적인 작업이 없고 argument를 받을 수 없으므로 현재 구현된 dataPost.cgi나 dataGet.cgi의 출력 내용을 web client에게 반환하게 된다.

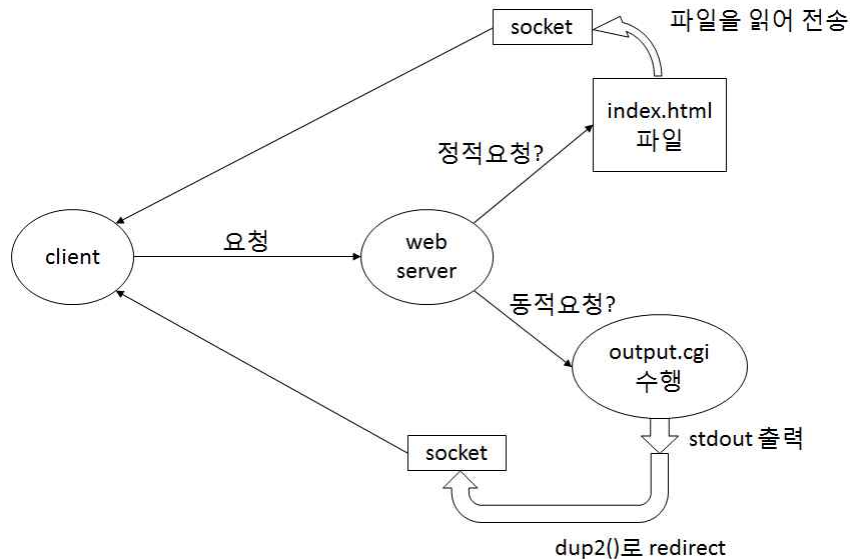


그림 13 Web server의 요청 처리 개요

- 동적 요청 기능을 구현하려면 우선 client로부터 받은 요청이 동적 요청인지 정적 요청인지 구분할 수 있어야 한다. Client로부터 들어온 요청들은 대부분 파일 확장자를 통해 정적 요청인지 동적 요청인지 구분할 수 있다.
  - 본 과제에서는 확장자가 .cgi인 경우에만 동적 요청으로 판단하고, 나머지의 경우 (.html, .htm, .txt 등) 정적 요청으로 판단한다.
- 파일명 분리
  - 동적 요청 여부를 request.c 파일의 parseURI()에서 확인하는데, GET method의 경우에는 파일명과 argument가 같이 붙어 있으므로 이를 분리해야 한다. parseURI() 함수의 else 부분이 파일명과 argument를 분리하는 부분이다. 앞서서도 언급했지만 server가 받은 uri가 “/output.cgi?data=100&cnt=5”이면 이를 “./output.cgi”와 “data=100&cnt=5”로 나누어 각각 filename과 cgiargs에 할당한다.
    - 슬래시 앞에 ‘.’이 붙은 것은 web 주소내의 경로를 server file system 내의 경로로 바꾼 것이다. 원래는 사용자의 계정에 따른 변환과정이 필요하지만 여기서는 간단하게 server 실행파일이 있는 디렉터리로 변경하였다.
- CGI 프로그램 실행
  - 동적 요청을 처리한다는 것은 server에 위치한 CGI 프로그램의 수행을 client가 요청하면, 그 프로그램을 Fork()와 Execve() 함수를 통해 실행하고, 그 결과를 client에게 돌려준다는 것을 의미한다. 이번 단계에서 server는 그 프로그램이 종료하기를 기다렸다 다음 요청을 소켓을 통해 받도록 구현한다.

- Execve(filename, argv, envp)는 문자열 filename에 있는 실행 file을 수행시키며 그 file의 command line argument로 argv를 전달하고 environment variable로 envp를 전달한다. 예를 들어, 실행 file명이 “./dataPut.cgi”이고 argv와 envp가 {NULL}이면 dataPut.cgi가 수행되지만 아무런 parameter도 전달되지 않는다.
- 관련함수
  - Fork(): stems.c에 정의되어 있다.
  - Execve(filename, argv, envp): stems.c에 정의되어 있다. argv나 envp에 전달할 내용이 없으면 이들을 {NULL}로 선언하면 된다. 그냥 NULL을 사용하면 Linux 배포판에 따라 에러가 발생할 수 있다.
  - Dup2(fildes, fildes2): fildes와 fildes2는 파일 기술자이다. 파일 기술자는 커널 내에 있는 현재 프로세스의 오픈된 파일들의 정보를 담고 있는 배열형 자료구조의 인덱스이다. dup2() 시스템 콜을 수행하면 그 자료구조의 fildes번째에 있는 내용이 fildes2번째에 덮어쓰기가 된다. 이 시스템 콜이 수행된 후 write(fildes2, buff, size)를 수행하면 실제로는 write(fildes, buff, size)를 한 것과 같은 결과가 발생한다.
- 체크 포인트 (보고서에 이에 대한 설명이 있어야 한다)
  - Web client는 dataPost.cgi나 dataGet.cgi의 결과를 제대로 받았는지를 설명하시오. (수행 결과를 스냅샷으로 찍고, 어째서 이 스냅샷이 프로그램이 제대로 수행되고 있음을 보여주는가를 설명할 것)
  - 동적 요청과 정적 요청을 어떻게 구분하는지 설명하시오.
  - Fork(), Execve(), Dup2()의 순서를 어떻게 해야 할 것인가?
    - 예를 들어, Dup2()가 Fork()보다 먼저 실행되면 CGI 프로그램의 stdout이 바뀌는 게 아니라 web server의 stdout이 바뀐다.
    - 순서를 정했으면 그 순서로 프로세스들이 수행될 때 어떻게 동작하는 지를 묘사하시오.

#### 4.2. Web server가 CGI 프로그램에게 argument 전달

- Web server의 소스코드를 변경하여 client로부터 받은 argument를 CGI 프로그램에게 전달하도록 하시오.
  - C 언어에서 부모 프로세스가 자식 프로세스에게 정보를 전달할 경로는 여러 개가 존재한다. 하지만, web server와 CGI 프로그램 간에는 환경변수를 이용하여 argument들을 전달하는 관행이 있다.
    - 이 관행을 지키면 다른 CGI 프로그램의 실행파일을 그대로 쓸 수 있다는 장점이 있다.
  - Web server는 client로부터의 query 종류에 따라 REQUEST\_METHOD라는 환경 변수에 “GET” 또는 “POST”를 할당한다.
    - Setenv() 함수를 사용하여 특정 환경변수 이름에 문자열을 할당한다. 예를 들어, Setenv(“REQUEST\_METHOD”, “GET”, 1) 이라고 하면 REQUEST\_METHOD라는 환경 변수에 문자열 “GET”을 할당한다.
  - Web server는 GET 요청에 대해서는 환경변수 QUERY\_STRING에 argument들을 문자열의 형태로 저장한다.

- 그림 6의 경우라면 “data=test&value=100”가 QUERY\_STRING에 할당된다.
- o Web server는 POST 요청에 대해서는 우선 환경변수 CONTENT\_LENGTH에 request body의 크기를 문자열의 형태로 저장된다.
- 예를 들어, 그림 9의 request를 받으면 web server는 request header와 빈 줄 (\r\n)까지 읽은 상태에서 남은 내용의 길이가 17이라는 것을 알게 된다. 이 “17”을 문자열의 형태로 환경변수 CONTENT\_LENGTH에 저장한다.
- o Server가 CGI 프로세스에게 환경변수를 통해 정보를 전달하는 경우, Execve()의 세 번째 parameter는 libC에서 전역변수로 사용하는 environ을 사용해야 한다.
- o Web server는 POST 요청에 대해서는 web client가 보내는 request body를 자신이 받는 대신 CGI 프로세스가 받아야 한다 (그림 14(a)). 이를 위해 Dup2() 함수를 이용하여 web client로부터 packet을 받는 server의 socket descriptor를 CGI 프로세스의 stdin (file descriptor로는 STDIN\_FILENO (unistd.h))에 덮어쓰는 작업을 진행해야 한다. 단계 1에서는 server에서 web client로 보내는 socket descriptor를 STDOUT\_FILENO에 덮어쓰는 과정을 진행했던 것을 기억할 것이다.

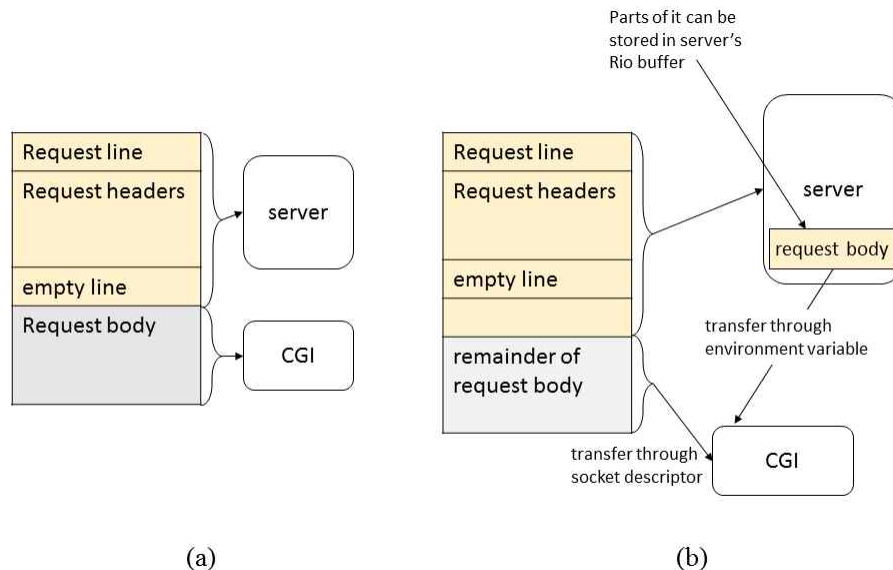


그림 14 HTTP 요청의 처리. (a) 각 프로세스가 web request를 받을 부분, (b) Robust-I/O로 인해 request body 일부가 server에 저장된 상태

- 여기서 주의할 점은 그림 14(b)처럼 Robust-I/O package가 packet을 읽을 때 buffering을 하기 때문에 request body까지 미리 읽어 Robust-I/O의 buffer에 저장해버렸을 가능성이 많다는 것이다. 이 문제를 해결하기 위해서 다음과 같은 방법을 사용할 수 있다.
  - server가 request의 empty line까지 읽은 상태라고 하자.
  - Rio\_readrestb() 함수를 사용하여 Robust-I/O의 buffer에 남은 내용을 모두 읽어 새로운 환경 변수에 저장한다. 이 때 환경 변수 이름은 기존 환경 변수와 겹치지 않는 임의의 이름을 사용해야 한다. (예를 들어 REST\_CONTENTS)
  - server가 CGI process를 실행한다.
  - CGI process가 그 환경 변수를 통해 그 request body를 받는다.

- 또는 stems.h에 있는 rio 구조체의 rio\_buf의 크기를 1로 줄임으로써 미리 읽기를 없앨 수도 있다. 이 방법은 위 문제를 간단히 해결하는 반면 성능을 대폭 떨어트릴 수 있으며 side effect를 주의해야 한다.
- CGI 프로세스가 argument를 web server로부터 받도록 수정하시오.
  - 주어진 CGI 코드들은 argument들을 받을 준비가 되어있지 않기 때문에 server로부터 argument들을 받을 수 있도록 수정되어야 한다. CGI 코드는 다음과 같은 내용을 포함하도록 한다.
  - CGI 프로세스는 환경변수의 내용을 읽고 어떤 형태의 요청인지 파악한다.
    - CGI 프로세스는 getenv() 함수를 사용하여 환경변수 REQUEST\_METHOD로부터 문자열을 얻는다.
    - 이 문자열은 “GET” 또는 “POST”이므로 요청 형태를 알 수 있다.
  - GET 요청인 경우에는 환경변수 QUERY\_STRING에서 argument를 얻는다.
  - POST 요청인 경우에는
    - 환경변수 CONTENT\_LENGTH로부터 request body의 길이를 얻는다.
    - Web server가 client로부터 request body의 일부를 Robust-I/O의 buffer로 미리 읽었을 가능성이 있기 때문에 미리 정한 환경변수 (예를 들어 REST\_CONTENTS)로부터 이 내용을 읽는다.
    - 환경변수에서 읽은 request body의 양이 CONTENT\_LENGTH로부터 얻은 request body의 길이보다 작은 경우 STDIN\_FILENO file descriptor (또는 stdin)로부터 남은 양의 request body를 읽어 이미 읽은 request body의 뒷 부분에 추가한다.
      - Web server가 client로부터의 socket descriptor로 STDIN\_FILENO를 덮어썼기 때문에 read(STDIN\_FILENO, , )나 Rio\_readnb() 등의 함수를 사용하여 web client로부터의 request body를 읽을 수 있다. CGI 프로그램이 Rio\_readnb()를 이용하려면 STDIN\_FILENO를 Rio\_readinitb에서 등록해야 한다.
  - dataPost.cgi가 web server에서 전달받는 내용은 (sensor name, time, value)이다. 표 2의 경우라면 “name=dummy&time=1235713769&value=22.4”가 전달된다.
  - dataGet.cgi가 web server에서 전달받는 내용은 (cmd, n)이지만 아직 clientGet 프로세스가 보낼 명령이 없기 때문에 표 1의 경우와 같이 보낼 내용이 없다. CGI 프로그램에게 argument를 보내는 기능이 추가되면 config-cg.txt는 다음 표 3와 같이 변경되어야 한다.

형식	설명	예시
server IP addr.	web server의 IP 주소	127.0.0.1
server port no.	web server의 port 번호	8080
CGI program	수행될 dataGet.cgi의 경로와 argument	/dataGet.cgi?command=LIST&n=0

표 3 config-cg.txt의 내용 2

- 표 3과 같은 내용을 읽은 clientGet 프로세스는 web server에 다음과 같은 내용을 HTTP protocol로 보낸다.

```
GET /dataGet.cgi?command=LIST&n=0 HTTP/1.1\r\n
Host: 127.0.0.1\r\n
Connection: keep-alive\r\n
....
\r\n
```

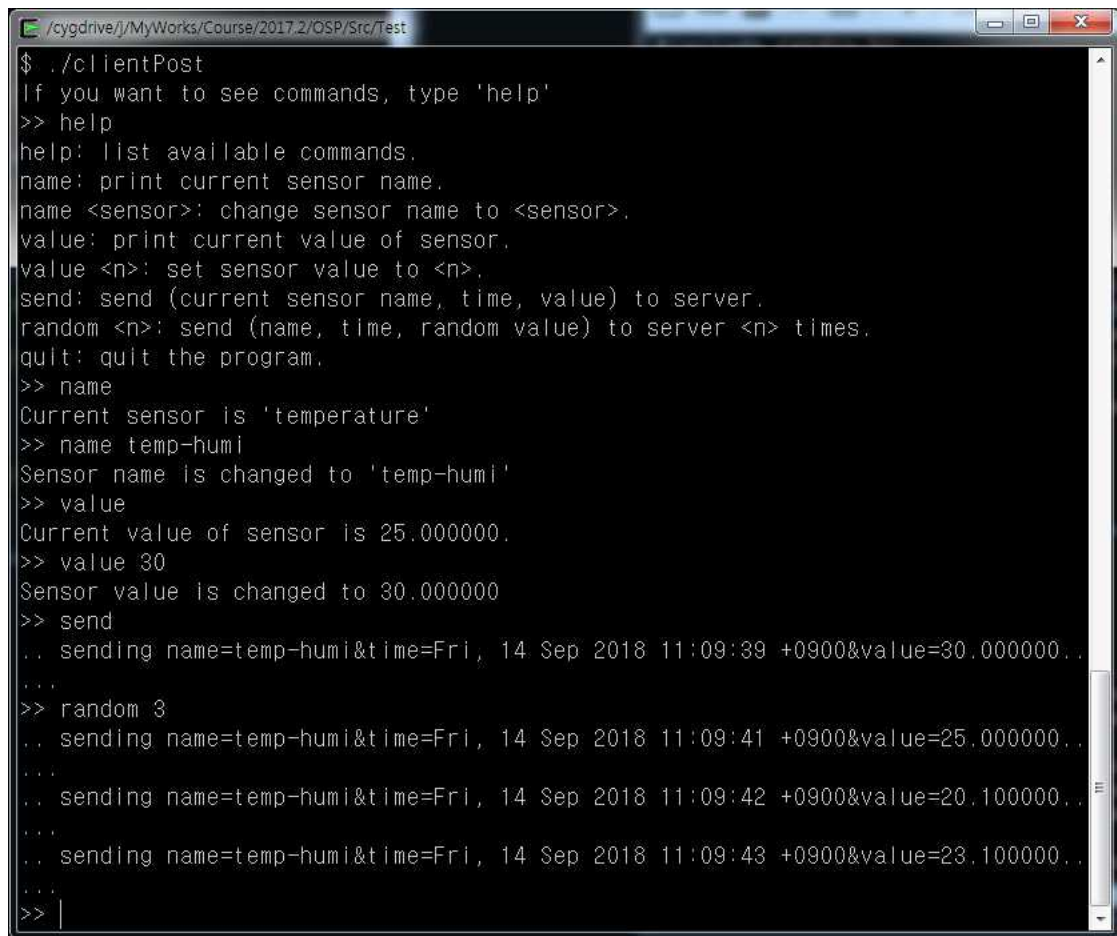
그림 15 표 3에 따른 clientGet이 web server에게 보내는 내용

- 그림 16과 같은 내용을 받은 web server는 “command=LIST&n=0”을 argument로서 dataGet.cgi에게 보낸다.
- 관련함수
  - Setenv(name, value, overwrite): stems.c에 정의되어 있다. name에는 환경 변수의 이름 value에는 그 환경 변수의 값인 문자열, overwrite는 그 환경 변수가 이미 있으면 새로운 값으로 덮어쓸지 여부를 결정한다. (0이면 덮어쓰지 않고, 1이면 덮어씀)
  - getenv(name): stdlib.h에 정의되어 있다. name에 해당하는 환경 변수의 내용을 가리키는 포인터를 반환한다.
- 체크포인트
  - Web request의 종류에 따라 사용해야 하는 환경변수의 명칭과 내용을 표로 정리하시오.
  - config-cg2.txt를 사용한 clientGet이 dataGet.cgi로부터 argument들을 제대로 돌려받았는가? (스냅샷 첨부와 설명)
  - 표 2의 config-cp.txt 내용을 configuration file로 사용한 clientPost가 제대로 argument들을 포함한 결과를 받았는가? (스냅샷 첨부와 설명)

#### 4.3. 콘솔로 제어하는 clientPost 구현

- Command shell의 형태로 clientPost가 사용자로부터 입력을 받고 web server로 request를 보내는 기능을 추가하시오. IoT 시스템에서 clientPost는 주기적으로 센서로부터 데이터를 읽어 server에 보내기 때문에 command shell과 같은 user interface가 필요하지 않다. 이번 단계에서 clientPost의 command shell을 구현하는 이유는 전체 IoT 시스템이 제대로 수행되는지를 판단하는 디버깅 기능을 추가하기 위해서이다. 그림 16은 clientPost가 수행되는 예시를 보여준다.





```
$ ./clientPost
If you want to see commands, type 'help'
>> help
help: list available commands.
name: print current sensor name.
name <sensor>: change sensor name to <sensor>.
value: print current value of sensor.
value <n>: set sensor value to <n>.
send: send (current sensor name, time, value) to server.
random <n>: send (name, time, random value) to server <n> times.
quit: quit the program.
>> name
Current sensor is 'temperature'
>> name temp-humi
Sensor name is changed to 'temp-humi'
>> value
Current value of sensor is 25.000000.
>> value 30
Sensor value is changed to 30.000000
>> send
.. sending name=temp-humi&time=Fri, 14 Sep 2018 11:09:39 +0900&value=30.000000..
...
>> random 3
.. sending name=temp-humi&time=Fri, 14 Sep 2018 11:09:41 +0900&value=25.000000..
...
.. sending name=temp-humi&time=Fri, 14 Sep 2018 11:09:42 +0900&value=20.100000..
...
.. sending name=temp-humi&time=Fri, 14 Sep 2018 11:09:43 +0900&value=23.100000..
...
>> |
```

그림 16 clientPost의 shell 기능 예제

- 추가할 shell 기능에는 다음과같이 기본적으로 5종류 (모두 7종류)의 내장 명령어들이 있다.
  - help: 내장 명령어들의 목록과 간단한 설명 출력
  - name: 현재 센서의 이름을 출력한다. clientPost는 1개의 센서만 존재한다고 보지만 테스트를 위해 다른 센서로 교환해서 사용하는 상황을 고려한 명령어이다.
  - name <sensor>: 센서의 이름을 <sensor>로 바꾼다.
  - value: 센서가 읽은 값을 보여준다. 실제 센서가 있는 것이 아니므로 이전에 설정한 값을 출력할 뿐이다.
  - value <n>: 센서 값을 <n>으로 변경한다. Threshold 보다 작거나 큰 값으로 설정함으로써 push alarm이 제대로 동작하는지 확인할 수 있다.
  - send: 현재 센서의 (이름, 시각, 값)을 web server에 HTTP request의 형태로 보낸다. 그 결과를 받은 후 화면에 출력한다. 그림 16은 결과를 받아 출력하는 부분이 ...으로 생략되어 있다.
  - random <n>: 현재 센서 값을 중심으로 -10 ~ +10 사이의 값을 임의로 선택해 1초마다 요청을 보내고 그 결과를 화면에 출력한다. 이 명령은 서버와 클라이언트가 연속된 요청을 무리없이 처리하는지를 확인하는데 사용할 수 있다.
  - quit: clientPost 프로그램을 종료한다.

- config-cp.txt는 표 4와 같은 내용을 가지도록 변경한다. 이 표에서 읽은 name과 value가 default sensor name과 value가 된다.

형식	설명	예시
server ip addr.	web server의 IP 주소	127.0.0.1
server port no.	web server의 port 번호	8080
CGI program	수행될 cgi program의 경로와 이름	/dataPost.cgi
name	sensor device name	temperature
value	sensor value 기본 값	15.0

표 4 shell 기능이 추가된 clientPost를 위한 config-cp.txt 내용

- 체크 포인트
  - clientPost가 그림 16과 같은 식으로 수행되는지 확인한다. 특히 send와 random 명령을 사용했을 때 web server로부터 받은 값들이 제대로 출력되는지 확인한다.

#### 4.4. CGI 프로그램과 DB 연동

- Server가 수행되는 컴퓨터에 DB 프로그램을 설치
  - 프로그램을 개발하고 있는 Linux에 설치할 수 있는 DB 프로그램을 설치한다.
    - MySQL이나 Cassandra를 권장한다.
    - Cassandra의 경우 C에서 사용할 수 있게 driver를 설치한다.
  - 필요한 table을 구성한다. 각 table은 다음과 같은 필드를 가진다. 필요한 필드를 추가할 수 있다. 그림 17은 본 과제에서 구축하는 DB table의 예제를 보여준다.
    - sensorList table: (name, id, count, average)
      - sensor client들의 목록이며 sensor table의 목록이기도 하다.
      - name: sensor들을 구분하는 문자열
      - id: 1부터 시작하는 sensor의 번호
      - count: 각 sensor data가 저장된 개수, 마지막 데이터를 찾는데 사용한다.
      - average: 저장된 sensor data의 평균값
    - sensor# table: (time, data, seq.num)
      - #는 번호로서 한 sensor의 id가 3이면 이는 3된다.
      - time과 data는 client로부터 받은 데이터
      - seq.num은 0부터 시작하는 저장된 data의 번호. 마지막 entry의 seq.num은 그 sensor의 count값과 같다.

sensorList				sensor1			sensor2		
name	id	cnt	ave	time	value	idx	time	value	idx
temperature	1	2	21.0	1535713769	20.0	1	1535713772	70.0	1
humidity	2	1	70.0	1535713779	22.0	2			

그림 17 DB table의 예제

- dataPost.cgi 프로그램은 환경변수를 통해 전달받은 argument 내의 정보를 DB에 저장
  - Query language를 사용하여 DB에 데이터를 저장하고 client에 회신

- 데이터 저장에는 sensorList table과 sensor# table을 모두 필요로 한다.
- 예를 들어, "name=temperature&time=1535713794&value=24.0"을 받았다면 그림 17의 DB table들은 그림 18와 같이 변경된다.

sensorList				sensor1			sensor2		
name	id	cnt	ave	time	value	idx	time	value	idx
temperature	1	3	22.0	1535713769	20.0	1			
humidity	2	1	70.0	1535713779	22.0	2	1535713772	70.0	1
				1535713794	24.0	3			

그림 18 DB table의 예제 2

- 특별한 문제가 없을 경우 200 회신을 수행한다.
- 체크 포인트
  - 아직 이름이 등록되지 않은 sensor에서 첫 데이터가 도착하면 어떤 작업을 수행해야 하는가? DB table의 예제 그림을 갱신하고, 이를 위한 동작을 제시하시오. 또한 구현된 결과를 스냅샷으로 입증하시오.
  - clientPost에서 5번 이상 send 명령 (또는 1번의 random 5 명령)을 수행한 뒤, DB의 셀을 사용하여 보낸 데이터들이 모두 저장되었는지 확인한다. 수행된 결과를 스냅샷으로 추가하고 정상 실행 여부를 설명하시오.

#### 4.5. clientGet 프로세스는 shell 형태의 CLI 프로그램으로 확장

- clientGet을 수행하면 prompt가 나타나고 사용자는 다음과 같은 명령어들을 수행하도록 셸 프로그램으로 확장하시오. 명령어들은 대문자나 소문자를 모두 처리하도록 하시오.
  - LIST: sensor들의 목록 출력
    - URI를 /dataGet.cgi?command=LIST 와 같은 형태로 전송
    - Web server측에 있는 DB의 sensorList table로부터 (name) 목록을 출력
  - INFO <sname>: sensor <sname>의 정보 출력
    - URI를 /dataGet.cgi?command=INFO&value=sname 과 같은 형태로 전송
    - DB의 sensorList table로부터 sname값을 가진 entry의 (count, average)을 얻어 화면에 (sname, count, average)을 출력. 해당 이름의 entry가 없는 경우 "There is no sensor named 'sname'." 출력
  - GET <sname>: sensor <sname>의 가장 최근 (time, value) 출력
    - clientGet 프로세스는 URI를 /dataGet.cgi?NAME=sname&N=1 과 같은 형태로 전송
      - 여기서 두 번째 argument 1은 가장 최근 data 1개를 얻겠다는 것을 의미한다.
    - Web server는 DB의 sensorList table로부터 sname에 해당하는 sensor의 id와 count를 얻는다.
    - Web server는 sensor# table로부터 count와 같은 seq.num을 가지는 entry의 (time, value) 값을 추출하여 clientGet에게 반환한다.
      - 예를 들어, 여기서 id가 3이면 sensor#는 sensor3 이 된다.
    - clientGet 프로세스는 server로부터 받은 (time, data)를 출력한다. 시간은 "Wed Sept 12 21:49:08 2018"과 같은 형식으로 출력해야 하며 ctime()과 같은 함수가 이리

한 변환을 지원한다.

- GET <sname> <n>: sensor <sname> 의 가장 최근 (time, value)를 n개 시간순서대로 출력
  - clientGet 프로세스는 URI를 /dataGet.cgi?NAME=sname&N=10 과 같은 형태로 전송 (n이 10인 경우)
  - 나머지는 GET 명령과 동일하다.
- quit: clientGet 프로세스를 종료한다.
- exit: clientGet 프로세스를 종료한다.
- dataGet.cgi 프로그램은 argument를 바탕으로 DB에서 정보를 읽어 client에 회신
  - 이 프로그램에게 요청되는 정보는 다음과 같다.
    - LIST: sensorList table에 있는 sensor들의 name들을 반환한다.
      - 그림 17의 경우에는 [temperature](#)와 [humidity](#)가 출력된다.
    - INFO s-name: sensor 's-name'의 데이터 개수 (count), 최근 데이터 값을 반환한다.
      - 그림 17의 경우에 INFO temperature를 입력했을 때, [2, 21.0](#)이 출력된다.
    - GET s-name: sensor 's-name'의 가장 최근 (time, data)를 반환한다.
      - 그림 17의 경우에 GET temperature를 입력했을 때, [Wed Sept 12 21:49:08 2018, 22.0](#)과 같은 문자열이 출력된다.
    - GET s-name \$\$: sensor 's-name'의 가장 최근 \$\$개의 (time, data)를 반환한다.
      - 그림 17의 경우에 GET temperature 10을 입력했을 때, [160.2, 22.0\n100.1 20.0](#)이 출력된다. 저장된 데이터가 부족하므로 10개의 data가 모두 출력되지 못했다.
  - clientGet 프로그램의 config-cg.txt 파일은 그림 19와 같이 서버 ip 주소와 port 번호만 들어가도록 변경한다. 하지만 이전 config-cg.txt를 사용하도록 코드를 유지해도 상관없다.

127.0.0.1 8080
-------------------

그림 19 5단계를 위해 변경된 config-cg.txt

- 그림 20은 그림 18과 같이 DB가 구성된 상태에서 각 명령어를 입력한 예상 결과를 보여준다.

```

$ ./clientGet
# LIST
temperature  humidity
# INFO temperature
count  averate
3      22.0
# GET temperature
Wed Sept 12 21:49:08 2018 22.0
# GET temperature 2
Wed Sept 12 21:49:08 2018 22.0
Wed Sept 12 21:49:23 2018 24.0
# quit
$

```

그림 20 push client가 push server에게 보내는 HTTP protocol

#### 4.6. Push alarm 기능 구현

- 기본적으로 server는 수동적이고 client는 능동적이다. 하지만 폭염 경보와 같은 경고 메시지를 보내야 하는 경우에는 server는 능동적으로, client는 수동적으로 동작해야 한다. 이러한 문제를 해결하는 방법들이 다음과 같이 3 가지가 있다.
  - **Alarm server**: web client에 alarm server를 두고 web server에 alarm client를 수행
  - **주기적 client**: web client에서 주기적으로 web server에 요청을 보내 alarm을 발생시켜야 할 상황이 발생했는지 확인
  - **주기적 multi-thread**: clientGet에 사용자의 명령을 보내는 기존 thread에 주기적으로 alarm이 발생했는지 요청하는 thread가 추가된다.

##### 4.5.1. Alarm server

- Alarm server의 기본 구조는 그림 19과 같이 사용자 측에 알람 서버를 두고 서버 측에 알람 클라이언트를 두는 것이다. 스마트폰에서 사용하는 push alarm은 별도의 알람 호스트 (본 과제의 클라이언트 역할)와 스마트폰의 서비스 (본 과제의 서버 역할)를 통해 동작하지만, 본 과제에서는 push alarm 기능을 구현하기 위해 작은 alarm client와 alarm server를 만들어서 사용한다. Alarm client는 web server측에 존재하고 alarm server는 사용자 측에 존재한다. 여기서 사용자는 DB에 저장된 내용을 보려는 clientGet 프로그램을 수행시키는 사람을 의미한다. Alarm server의 구조는 web server의 구조와 유사하고 alarm client도 마찬가지이다.

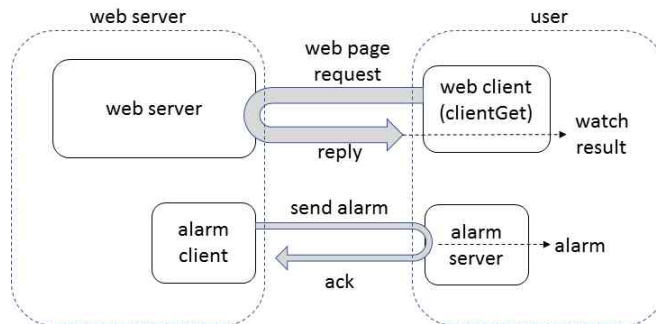


그림 21 Push alarm이 추가된 시스템

- dataPost.cgi 변경
  - 지금까지 dataPost.cgi는 server에서 받은 내용을 DB에 저장하는 기능만 있었다. Push alarm이 있는 시스템에서는 그림 20과 같이 dataPost.cgi가 alarm client에게 경고를 알려야 할 내용을 named pipe를 통해 alarm client에게 보내는 코드가 추가되어야 한다. 관련된 named pipe 함수는 다음과 같다.
    - Named pipe의 생성: mkfifo()
    - 생성된 named pipe의 open: open()
    - Named pipe를 통해 값을 보내고 받기: write(), read()
    - Named pipe를 더 이상 사용하지 않을 때: unlink()

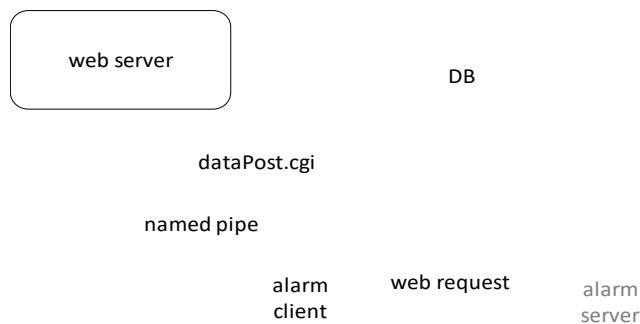


그림 22 dataPost.cgi와 alarm client와의 관계

- 예를 들어 dataPost.cgi가 argument “name=temperature&time=1535713794&value=31.5”를 web server로부터 받게 되면 이를 DB table에 반영하여 그림 18과 같이 DB를 변경하고, alarm client에게도 named pipe를 통해 이 문자열을 전송한다.
- 한 가지 주의할 점은 이 문자열을 받을 alarm client는 그 길이를 알지 못한다는 문제가 있다. 이에 대한 해결책의 하나로 우선 정수 변수 크기로 문자열의 길이를 보내주고 다음에 이 문자열을 보내는 방법을 생각할 수 있다.
- Alarm client 프로세스는 다음과 같이 진행된다.
  - Web server가 fork와 exec() 계열 함수를 통해 alarm client를 생성한다.
  - Alarm client는 config-pc.txt 파일을 읽는다. 그 내용은 표 5와 같다.

형식	설명	예시
server ip addr.	alarm server의 IP 주소	127.0.0.1
server port no.	alarm server의 port 번호	9090
CGI program	수행될 alarm.cgi의 경로와 이름	/alarm.cgi
threshold	이 값을 넘는 데이터가 들어오면 alarm server에 알람을 전송	27

표 5 config-pc.txt의 내용

- Named pipe를 통해 dataPost.cgi로부터 argument를 받는다. Argument의 길이가 매번 바뀐다는 점을 주의한다.
- 받은 값들을 분리하여 문자열과 정수 또는 실수로 구분한다. 예를 들어, 받은 argument가 “name=temperature&time=1535713794&value=31.5”라면 문자열 “temperature” (name), 문자열 “1535713794” (time), 그리고 실수 31.5 (value)로 나누어진다. 여기서 value 31.5가 threshold 27을 넘으므로 push server에게 POST method의 web request를 보낸다.
- 이 경우 그림 14와 같은 내용으로 push server에게 전송된다.

```
POST /alarm.cgi HTTP/1.1\r\n
Host: 127.0.0.1:9090\r\n
Connection: keep-alive\r\n
Content-Length: 43\r\n
...
\r\n
name=temperature&time=1535713794&value=31.5
```

그림 23 push client가 push server에게 보내는 HTTP protocol

- 요약하자면 다음과 같은 과정이 반복 수행된다.
  - Named pipe로부터 data를 받는다.
  - data가 threshold를 넘으면 push server에게 web request를 보낸 후 응답을 받는다.
- Push server
  - clientGet 프로세스의 자식 프로세스로서 push client가 보낸 요청을 받으면 alarm.cgi를 수행시킨다.
  - pushServer.c 파일로 구현하며 config-ps.txt 파일로부터 port number를 읽는다.
    - web server와 push server가 같은 컴퓨터에 있는 경우를 대비해서 이 파일의 port number는 다르게 설정할 필요가 있다. (예: 9090)
  - pushServer 프로세스는 다음과 같은 과정을 반복한다.
    - Port로부터 POST 요청을 기다린다.
    - 받은 POST 요청에는 URI로 alarm.cgi가 있으며 request body로 (name, time, value)가 있다.

- 예를 들어 temperature sensor로부터 31.5라는 값이 push alarm의 값으로 전달된 경우 “name=temperature&time=1535713794&value=31.5”가 request body의 내용이 된다.
- 환경변수를 이용해 (name, time, value)를 alarm.cgi에게 전달한다.
- Fork()와 Execve() 함수를 이용하여 alarm.cgi를 수행시킨다.
- pushServer 프로세스의 동작은 지금까지 구현한 web server의 동작과 일치한다. 따라서 pushServer.c는 server.c에서 config-ps.txt를 읽는 부분만 다르다고 보면 된다.
- alarm.cgi
  - 실제 push server측에서 push 역할을 담당하는 프로세스는 alarm.cgi로 다음과 같은 작업을 수행한다.
    - WARNING: (name, time, value)을 화면에 읽을 수 있는 형태로 출력한다. (stderr 사용)
      - 예: “경고: temperature sensor로부터 345.2 시각에 31.5라는 값이 발생했습니다.”
    - push client에게 200 응답을 보낸다.
    - alarm.cgi는 clientGet과는 별도의 통신을 하지 않는다.

#### 4.5.2. 주기적 client

- 주기적 client를 이용하여 push alarm을 구현할 때는 반복적으로 요청을 보내는 clientGet 프로그램과 이를 고려한 dataPost.cgi와 dataGet.cgi를 만들어야 한다. 일반적으로 CLI는 사용자 요청을 계속 기다리지만 이 방법의 clientGet은 무한히 사용자의 입력을 기다리지 않는다. 서버에서 알람에 해당하는 이벤트가 발생했을 가능성 때문이다. 따라서 clientGet은 timeout을 정해두고 그 시간 동안만 사용자 입력을 기다린다. scanf같은 입력 명령은 timeout 기능이 없지만 select() 함수를 사용하여 timeout 기능을 추가할 수 있다. select() 함수의 사용 예는 부록에 제시되어 있다.
- clientGet이 명령어를 입력받은 경우에는 web server에 요청하고 그 응답을 기다렸다가 그 내용을 화면에 출력하고 다시 명령어를 기다린다. 화면에 출력하는 내용은 사용자가 쉽게 인식할 수 있는 문장으로 변경되어 있어야 한다. 예를 들어, “Warning: the temperature is 31.5 degree at 11 Sept. 2018 13:00:01”과 같은 식이다. 온도가 Timeout 동안 명령어를 받지 못한 경우에는 web server에 ‘alarm’이라는 명령어를 argument로 담은 요청을 보낸다. 그 응답에는 아무런 내용이 없을 수도 있고 특정 메시지가 있을 수도 있다. 어떤 경우든 그 내용을 화면에 출력하고, prompt를 출력한 후 다시 사용자의 입력을 기다린다. 이 경우 config-cg.txt에는 timeout 기간이 명시되어야 한다.
- dataGet.cgi는 이전의 명령어에 alarm이라는 명령어가 추가된다. DB에 alarm을 위한 테이블이 있으며 dataPost.cgi가 alarm에 해당하는 data가 입력된 경우 경고 메시지를 저장하는 테이블이다. 주기적 client로 push alarm을 구현하는 경우 이 테이블을 DB에서 관리해야 한다. dataGet.cgi는 alarm이라는 명령어가 포함된 요청을 받으면 이 alarm 테이블을 내용을 모두 읽어 응답 메시지에 넣어 보낸다. 내용을 모두 읽은 alarm 테이블은 그 내용을 비운다.
- dataPost.cgi는 자신이 실행되면서 받은 sensor 값이 threshold를 넘으면 DB의



alarm 테이블에 경고 메시지를 저장한다. 그 메시지는 센서에 따라 다르지만 (이름, 시각, 값)을 담고 있다. 예를 들어, “name=temperature&time=1535713794&value=31.5”와 같은 내용이다.

#### 4.5.3. 주기적 multi-thread

- 기존 clientGet은 사용자로부터 명령을 받고 server에 명령을 요청하고 그 결과를 받아 화면에 출력하는 일을 반복한다. 여기에 추가 thread를 두어 alarm 메시지 확인을 전담하게 한다. clientGet이 실행되면 alarm thread를 하나 생성하고 원래 작업으로 돌아간다. alarm thread는 server에 ‘alarm’ 명령어가 포함된 요청을 보낸다. Web server는 주기적 client와 동일하게 동작한다. 응답을 받은 alarm thread는 그 내용을 가공하여 화면에 출력하고 다시 일정 시각을 대기한다. 대기하는 시각은 config-cg.txt에 명시되어야 한다.
- clientGet이 종료될 때 alarm thread도 종료되어야 한다.
- 체크 포인트
  - 세 방법들 중 한 가지 방법을 선택하여 구현할 텐데, 그 방법을 선택한 이유와 근거를 구체적으로 기술하시오.
  - 선택한 방법을 구현하여 실행한 결과 어떤 문제점이 있는지 자세하게 기술하시오.
  - 주기적 multi-thread 방식으로 구현했을 때 alarm thread를 어떻게 종료시켰는지 구체적으로 기술하시오.
  - 더 나은 push alarm 방법이 있다고 판단되면 그 구조를 구체적으로 명시하고 어떤 점에서 장점이 되는지 설명하시오.
  - 실행 스냅샷을 통해 프로그램이 정상적으로 실행되고 있음을 입증하시오.

#### 4.7. 라즈베리파이에서 온습도 데이터를 주기적으로 서버로 POST

- 2단계가 완료된 clientPost를 라즈베리파이에서 수행되도록 변경하고 온습도 센서를 읽어 web server로 보내는 clientRPI.c 코드 구현
  - 라즈베리파이 개발 환경 설정: LMS 학습하기 참고
  - clientPost.c의 라즈베리파이 포팅: Raspberry PI의 일반적 운영체제인 Raspbian은 Debian 계통의 Linux이므로 POSIX 표준을 따르는 이 소스코드는 특별한 변경 없이 바로 사용할 수 있다.
  - 온습도 센서값 읽기
    - WiringPi library를 설치 (<http://wiringpi.com/>)
      - <http://www.rasplay.org/?p=3241> 참고 (2. 개발환경)
    - 그림 24과 같이 센서의 pin을 GPIO의 해당 pin과 연결 (GPIO## 이름의 어떤 핀과도 연결할 수 있지만 그 번호는 기억해야 한다.)
      - <http://eigerc.blogspot.kr/2015/01/raspberry-pi-dht11.html> 참고
      - [http://mechasolutionwiki.com/index.php?title=DHT11\(온습도\\_센서\)\\_사용하기\\_\(C\)](http://mechasolutionwiki.com/index.php?title=DHT11(온습도_센서)_사용하기_(C)) 참고

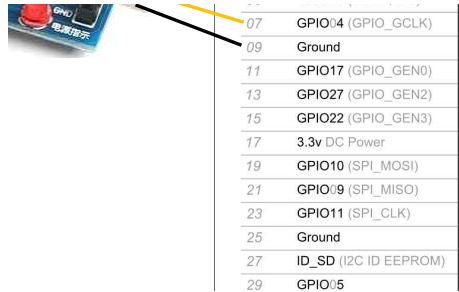


그림 24 온습도 sensor와 라즈베리파이 GPIO의 연결

- 온습도 센서로부터 값을 읽는 예제는 부록의 dht11exam.c 참고
  - Dht11 센서는 그림 28에 나온 시그널에 따라 초기화되고 값을 전달한다. 첫 부분 (빨간색 선)은 센서를 활성화시키기 위해 라즈베리파이가 시그널을 보내는 부분이고 다음 부분 (파란색 선)부터 센서로부터 시그널이 온다. 첫 번째 시그널은 시그널이 시작된다는 신호이므로 무시하고 두 번째부터 데이터를 받는다.
  - 5 byte를 받은 경우, 0번째와 1번째는 습도의 정수 부분과 소수 부분, 2번째와 3번째는 온도의 정수 부분과 소수 부분, 마지막 4번째는 패리티이다. (0번째부터 3번째까지의 값을 더하고 0xFF보다 큰 부분을 제거한 값이 4번째 byte와 같으면 데이터가 제대로 도착했다는 것을 의미한다.)
  - 예를 들어, 받은 값이 [40, 2, 27, 0, 69]이면 데이터가 맞고, 습도는 40.2%, 온도는 27.0을 의미한다.
- config-pi.txt의 내용을 아래 표와 같이 설정 (라즈베리파이와 web server가 서로 네트워크에 접속되어 있어야 하며 web server는 라즈베리파이에서 접근할 수 있어야 한다.)

형식	설명	예시
server ip addr.	web server의 IP 주소	192.168.100.200
server port no.	web server의 port 번호	8080
CGI program	수행될 cgi program의 경로와 이름	/dataPost.cgi
period	data를 보내는 시간 간격 (단위: 초)	20

표 6 config-pi.txt 내용

- clientRPI 프로세스는 수행초기에 config-pi.txt 파일로부터 server 및 전송 주기 정보를 읽는다.

- 예를 들어, 부록의 dht11exam.c는 전송 주기가 1초로 설정되어 있다. (delay(1000))
- 센서가 바르지 않은 데이터를 반환하는 경우도 있으므로 이러한 경우에 대비할 필요가 있다.
- 온도와 습도 데이터는 한 온습도센서에서 동시에 읽지만 web server에는 따로따로 전송한다. 온도 데이터를 보내고 그 응답을 web server로부터 받은 후 period/2 초만큼 기다린다. 다음 저장된 습도 데이터를 보낸다. 역시 period/2 초만큼 기다린 후 온/습도 데이터를 센서로부터 읽고 다음 온도 데이터를 보낸다. 라즈베리파이의 온도 센서의 이름은 temperaturePI, 습도 센서의 이름은 humidityPI로 지정한다. 다음은 각 센서의 정보를 서버에 보내는 예제를 보여준다.
  - 온도를 보낼 때는 name=temperaturePI&time=1535713794&value=31.1 형태의 request body를 전송
  - 습도를 보낼 때는 name=humidityPI&time=1535713804&value=54.2 형태의 request body를 전송
- 라즈베리파이는 자체 실시간 시계를 가지고 있지 않기 때문에 프로그램을 실행시키기 전 현재시각을 확인해야 한다.
- 체크포인트
  - clientGet에서 DB에 저장된 온습도계 기록내용을 읽어 최근 값들이 갱신되고 있는지 확인
  - config-pi.txt의 period를 5초 이내로 바꾸고, config-pc.txt의 threshold값을 적절하게 바꾸어 실행한 뒤 센서에 입김을 불어 push 경고가 clientGet에서 발생하는지 확인하고 이를 찍은 스냅샷을 첨부하여 시스템이 제대로 수행됨을 설명할 것

#### 4.8. Web server를 thread pool로 구현

- 주어진 web server는 single thread로 구현되어 있다. 하지만 single threaded server는 한 번에 하나의 HTTP 요청만 처리하기 때문에 성능이 낮다. 즉, 한 client에서 발생한 요청 (특히 파일 읽기 같은 시간이 오래 걸리는 요청)을 server가 처리하는 동안 다른 client에서 발생한 요청들은 모두 기다려야 한다.
  - 개선책: 고정된 크기의 worker thread pool을 만들어 사용 (이번 과제에서 구현)
    - Pool에 있는 thread들은 요청이 들어올 때까지 block된 상태로 대기
    - 요청들의 개수가 thread들의 개수보다 적으면 block된 thread들이 존재. 반대로 요청들의 개수가 thread들의 개수보다 많으면 그 요청들은 자신에게 thread가 할당될 때까지 대기
    - POSIX thread를 사용해야 하며 새로운 thread를 만드는 pthread\_create() 함수와 thread를 다루는 pthread\_detach() 함수를 익혀야 함
- 구현해야 할 내용
  - Web server는 config-ws.txt로부터 표 7과 같은 내용을 읽는다.

형식	설명	예시
server port no.	web server의 port 번호	8080
P	thread pool의 크기	10
N	queue의 크기	5

표 7 config-ws.txt 내용

- main()에 해당하는 master thread는 요청 정보들을 저장할 queue를 만든다.
- Master thread는 다음과 같은 과정을 거쳐 thread-pool을 구성해야 한다.
  1. config-ws.txt에서 제시된 thread pool의 크기 (P)만큼 thread들을 만든다.
  2. Socket을 통해 새로운 http 연결을 받으면 queue에 요청 정보 (그 socket의 기술자(descriptor)와 요청을 받은 시각)를 enqueue한다.
    - Master thread는 저장된 기술자로부터는 더 이상 통신을 하지 않는다.
  3. 저장 후 다음 http 연결을 기다리다 새로운 연결을 받으면 2로 이동한다.
- Worker thread의 진행 과정
  - 각 thread는 정적 요청이나 동적 요청을 처리할 수 있다고 본다.
  - 위에서 언급한 queue에 새로운 요청 정보가 enqueue되면 블록된 thread가 하나 깨어난다.
  - 깨어난 worker thread는 dequeue하여 요청 정보를 얻고 이 정보에 저장된 네트워크 기술자에 대해 읽기를 수행한다.
  - 요청 내용에 따라 정적 파일을 읽거나, CGI 프로세스를 수행한다.
  - 해당 네트워크 기술자에 그 결과를 기록해서 client에게 전송한다.
  - 다음 HTTP 요청이 자신에게 할당되기를 기다린다.
- Master thread와 worker thread사이의 관계
  - 기본적으로 producer-consumer 관계를 가진다. 그림 25에서와 같이 master thread (또는 main() 함수)는 accept() 시스템 콜을 통해 서버 포트 (listen port)를 통해 웹 요청을 기다리다 새로운 요청이 도착하면 그 요청에 대응하는 port 번호를 새로 부여받는다. Master thread는 그 port 번호를 queue에 넣고 다시 서버 포트를 통해 웹 요청을 기다리는 작업을 반복한다.
  - Master thread는 queue가 가득 찬 상태라면 다음 요청 정보를 넣기 전에 block되어야 하고, worker thread는 queue가 비어있으면 block되어야 한다. 즉, 운영체제 과목에서 다룬 bounded buffer 문제가 이 경우에 발생한다.
  - Semaphore는 bounded buffer 문제를 해결하기에 매우 효율적인 방법이므로 본 과제에서는 sem\_wait()나 sem\_post()와 같은 동기화 함수를 사용해야 한다. (busy waiting으로 구현하면 감점)

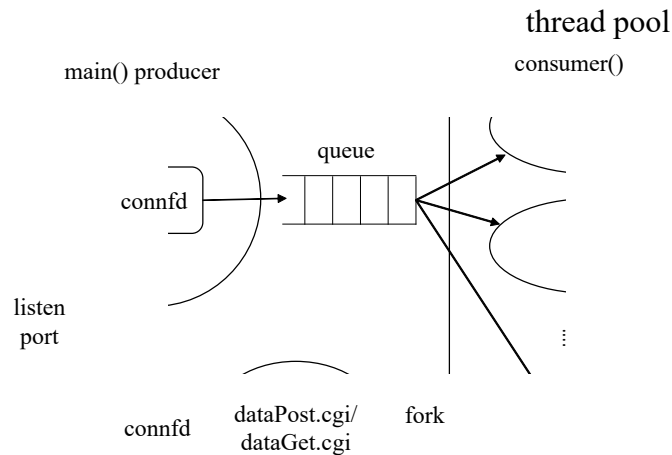


그림 25 thread pool

○ 주의사항

- CGI process를 수행하려면 fork 명령어로 새로운 프로세스를 생성해야 하지만 그렇다고 동시에 수행하는 thread의 개수가 증가하는 것은 아니다. 그 이유는 새로운 프로세스를 생성한 worker thread는 다른 일을 하지 않고 그 프로세스가 끝나기까지 블록되어 있어야하기 때문이다. 따라서 multi-thread 기능으로 구현할 때 CGI를 다루는 코드 부분은 변경할 필요가 없다.

#### 4.9. 프로그램 개선

- 다음과 같은 기능 추가를 할 수 있다. 이외에도 다양한 기능 추가가 가능하다.
  - 여러 개의 라즈베리파이와 센서를 이 시스템에 추가 (재료비 신청이 가능한 경우)
  - 스마트폰을 라즈베리파이와 센서 대용으로 사용
  - 다른 DB를 사용
  - clientGet 프로세스를 web page의 형태로 구현 (이 경우 push를 받기 위해 java나 javascript등을 사용할 필요가 있다.)
  - 작성한 web server 대신 Apache와 같은 일반적인 web server를 시스템에 적용
  - 구현한 clientGet 프로세스 대신 Google chrome과 같은 일반적인 web browser를 사용하도록 dataGet.cgi를 변경

#### 4.10. 추후 계획

- 최종 결과물에 어떤 기능을 더하여 공모전 등에 제출할 것인지 계획을 발표

## 5. 부록

- select() 함수 예제

```
<stdlib.h>
#include <stdio.h>
#include <sys/select.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

#define MAXBYTES 80

int main(int argc, char *argv[])
{
    fd_set readfds;
    int num_readable;
    struct timeval tv;
    int num_bytes;
    char buf[MAXBYTES];
    int fd_stdin;

    fd_stdin = fileno(stdin);

    while(1) {
        FD_ZERO(&readfds);
        FD_SET(fileno(stdin), &readfds);

        tv.tv_sec = 10;
        tv.tv_usec = 0;

        printf("Enter command: ");
        fflush(stdout);
        num_readable = select(fd_stdin + 1, &readfds, NULL, NULL, &tv);
        if (num_readable == -1) {
            fprintf(stderr, "\nError in select : %s\n", strerror(errno));
            exit(1);
        }
        if (num_readable == 0) {
            printf("\nPerforming default action after 10 seconds\n");
            break; /* since I don't want to test forever */
        } else {
            num_bytes = read(fd_stdin, buf, MAXBYTES);
            if (num_bytes < 0) {
                fprintf(stderr, "\nError on read : %s\n", strerror(errno));
                exit(1);
            }
            /* process command, maybe by sscanf */
            printf("\nRead %d bytes\n", num_bytes);
            break; /* to terminate loop, since I don't process anything */
        }
    }

    return 0;
}
```

- dht11 sensor device로부터 값을 얻는 코드 예제 (dht11exam.c)

```

#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#define MAXTIMINGS 83
#define DHTPIN 7
int dht11_dat[5] = {0, 0, 0, 0, 0};
void read_dht11_dat()
{
    uint8_t laststate = HIGH;
    uint8_t counter = 0;
    uint8_t j = 0, i;
    uint8_t flag = HIGH;
    uint8_t state = 0;
    float f;

    memset(dht11_dat, 0, sizeof(int) * 5);
    pinMode(DHTPIN, OUTPUT);
    digitalWrite(DHTPIN, LOW);
    delay(18);
    digitalWrite(DHTPIN, HIGH);
    delayMicroseconds(30);
    pinMode(DHTPIN, INPUT);
    for (i = 0; i < MAXTIMINGS; i++) {
        counter = 0;
        while (digitalRead(DHTPIN) == laststate) {
            counter++;
            delayMicroseconds(1);
            if (counter == 200) break;
        }
        laststate = digitalRead(DHTPIN);
        if (counter == 200) break; // if while broken by timer, break for
        if ((i >= 4) && (i % 2 == 0)) {
            dht11_dat[j / 8] <= 1;
            if (counter > 20) dht11_dat[j / 8] |= 1;
            j++;
        }
    }
    if ((j >= 40) && (dht11_dat[4] == ((dht11_dat[0] + dht11_dat[1] + dht11_dat[2] +
                                         dht11_dat[3]) & 0xff))) {
        printf("humidity = %d.%d %% Temperature = %d.%d *C \n", dht11_dat[0],
              dht11_dat[1], dht11_dat[2], dht11_dat[3]);
    }
    else printf("Data get failed\n");
}

int main(void)
{
    printf("dht11 Raspberry pi\n");
    if (wiringPiSetup() == -1) exit(1);
    while (1) {
        read_dht11_dat();
        delay(1000);
    }
    return 0;
}

```

그림 27 dht11 sensor 값을 읽는 코드 예제

- dht11의 시그널

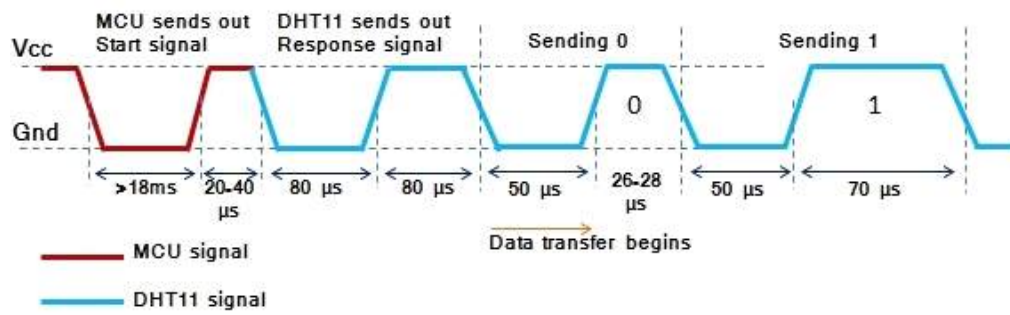


그림 28 dht signal