Embedded Systems

Prof. Myung-Eui Lee (F-102)
melee@kut.ac.kr
한국기술교육대학교
korea University of Technology and Education

Socket

- **Socket**
  - **An interface between application and network**
  - **The application creates a socket**
  - **The socket *type* dictates the style of communication**
    - reliable vs. best effort
    - connection-oriented vs. connectionless
  - **pass data to the socket for network transmission**
  - **receive data from the socket (transmitted through the network by some other host)**
- **Socket types**
  - ***stream sockets :* SOCK_STREAM**
    - Stream sockets treat communications as a continuous stream of characters
  - ***datagram sockets :* SOCK_DGRAM**
    - datagram sockets have to read entire messages at once.

Socket

- **Communications protocol**
  - Stream sockets use TCP (Transmission Control Protocol), which is a reliable, stream oriented protocol.
  - Datagram sockets use UDP (Unix Datagram Protocol), which is unreliable and message (fixed length) oriented.
- **STREAM**
  - a full-duplex communication channel between a user-level process and a device in Unix System V and beyond

SOCK_STREAM
TCP sockets
reliable delivery
in-order guaranteed
connection-oriented

SOCK_DGRAM
UDP sockets
unreliable delivery
no order guarantees
connectionless

Socket

- **int s = socket(int domain, int type, int protocol)**
  - **s: socket descriptor (like a file descriptor)**
  - **domain: communication domain**

    /usr/src/linux-2.4/include/linux/socket.h
    - AF_UNIX : Local Unix Domain protocol
    - AF_INET : IPv4 protocol
    - AF_INET6 : IPv6 protocol
    - AF_X25 : X25 packet protocol
    - AF_APPLETALK : AppleTalk protocol
  - **type: communication type**

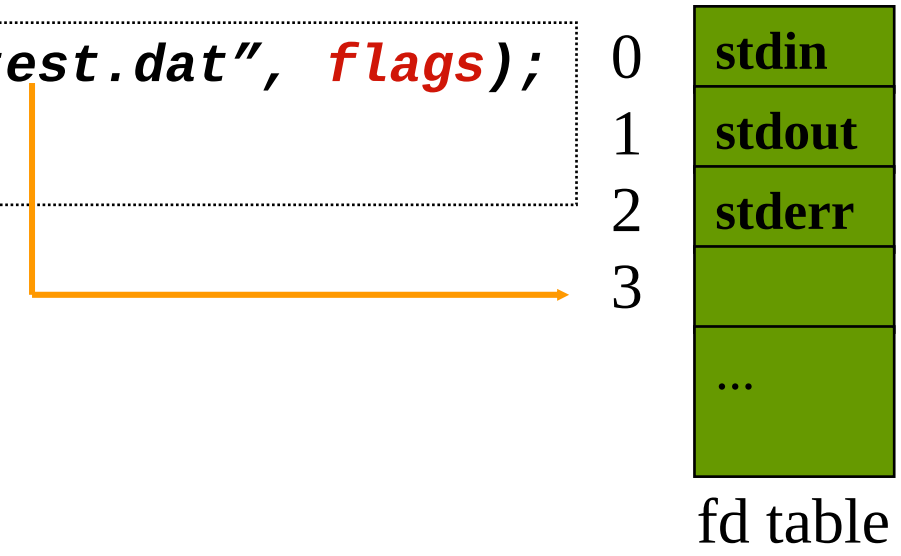
    /usr/src/linux-2.4/include/asm/socket.h
    - SOCK_STREAM: reliable, connection-based service
    - SOCK_DGRAM: unreliable, connectionless
  - **protocol: protocol family**
    - Specify a particular protocol to be used with the socket.
    - Normally only a single protocol exists to support a particular socket type, in which a case protocol can be specified as 0 (usually set to 0)

man socket

Socket

- **File Descriptor = Socket Descriptor**
  - » **a per process unique, nonnegative integer used to identify an open file**
  - » **file open**
    - search for the first empty slot in the process file descriptor table.
    - Allocate an open file description in the file table, which has a pointer to the inode table.
    - Flags : O_CREAT, O_APPEND, O_TRUNC, O_RDONLY, O_WRONLY, O_RDWR

```
int fd = open("test.dat", flags);
read(fd, …);
```



Socket

- **Example**
  - gcc –o fd-test fd-test.c
  - ./fd-test

```
#include <stdio.h>
#include <sys/socket.h>
#include <fcntl.h>

main()
{
      int fd1, fd2, fd3;

      fd1= socket(PF_INET, SOCK_STREAM, 0);
      fd2= open("test.dat", O_CREAT);
      fd3= socket(PF_INET, SOCK_STREAM, 0);

      printf("fd 1 (socket) : %d₩n", fd1);
      printf("fd 2 (file) : %d₩n", fd2);
      printf("fd 3 (socket) : %d₩n", fd3);

      close(fd1);
      close(fd2);
      close(fd3);
}
```

Network Device Driver

Network Device Layer

VFS layer	struct file_operations /* include/linux/fs.h */	
BSD socket layer	struct net_proto_family /* include/linux/net.h */	126
	…… struct socket	65
INET layer AF_INET	struct sock /* include/net/sock.h */	514
Transport layer	struct tcp_opt /* include/net/sock.h */	255
TCP/UDP	…….. struct proto	706
Network layer IP	struct tcp_func /* include/net/tcp.h */	555
Device layer	struct packet_type /* include/linux/netdevice.h */	447
	…… struct net_device	235, 450
	cs89x00.c /* drivers/net */	
	…….. net_open(struct net_device *dev)	1110

Socket

- **Socket address**
  - » **General address structure to support different address formats for different address family**
    - address family, port number, and a field for addresses of different sizes.

```
* include/linux/socket.h
    struct sockaddr
    {
            unsigned short sa_family;          /* Address family (AF_INET/IPv4) */
            char sa_data[14];                  /* Protocol-specific address information */
    };

* include/linux/in.h
    struct sockaddr_in
    {
            unsigned short sin_family;         /* Internet protocol (AF_INET) */
            unsigned short sin_port;           /* Port (16-bits) */
            struct in_addr sin_addr;           /* Internet address (32-bits) */
            char sin_zero[8];                  /*padding,  not used */
    };
    struct in_addr
    {
             unsigned long s_addr;             /* Internet address (32-bits) */
    }
```

Client/Server

- **The client server model**
  - Most interprocess communication uses the *client server model*.
  - These terms refer to the two processes which will be communicating with each other.
  - One of the two processes, the *client*, connects to the other process, the *server*, typically to make a request for information.
- **The steps involved in establishing a socket on the *client* side are as follows:**
  - Create a socket with the socket() system call
  - Connect the socket to the address of the server using the connect() system call
  - Send and receive data. There are a number of ways to do this, but the simplest is to use the read() and write() system calls.

Socket

- **The steps involved in establishing a socket on the *server* side are as follows:**
  - **Create a socket with the socket() system call**
  - **Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number and host address on the host machine.**
  - **Listen for connections with the listen() system call. Block until connection from client.**
  - **Accept a connection with the accept() system call. Send and receive data**
- **Socket Interface Call**
  - **socket() : create socket**
  - **bind() :**
    - on the server side for multiplexing
    - bind the socket to a specific port number for the listening socket

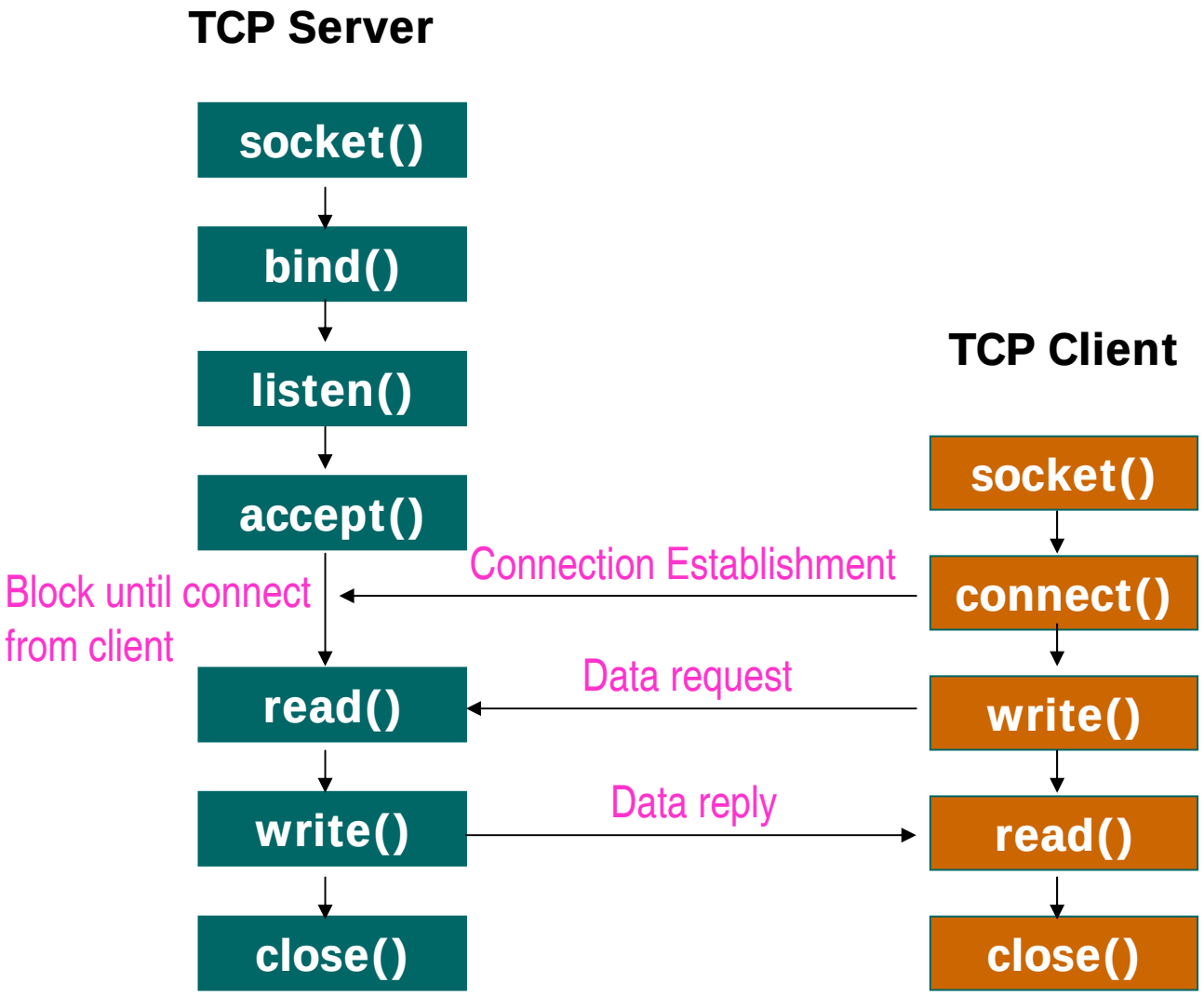
Socket

- **listen() :**
  - To avoid having protocols reject incoming request, a server may have to specify how many messages need to be queued
  - listen(socket, backlog); define the maximum length of the queue of pending connections
- **connect() :**
  - An application program should call connect to establish a connection before it can transfer data thru' reliable stream socket.
- **accept() :**
  - blocks until a connect calls the socket associated with this connection.
- **read(), recv(), recvfrom, recvmsg : Data read**
- **write(), send(), sendto, sendmsg : Data write**
- **close() : close socket**

man recv

man send

Socket (tcp)


TCP Server
socket()
bind()
listen()
accept()
Block until connect
from client
read()
write()
close()
TCP Client
socket()
connect()
write()
read()
close()
Connection Establishment
Data request
Data reply

Socket Program

- **Client : host – 192.168.1.100**
  - Download socket test program : microcom
  - make clean & make

```
CC          :=/opt/iwmmxt-1.0.0/bin/arm-linux-gcc
all:    server client
client: client.c
            gcc -o client client.c
server: server.c
            $(CC) -o server server.c
clean:
            rm client server
```

  - server : download to taret

- **Server : target -192.168.1.128**
  - ./server &

Socket Program

- **Client : host**
  - » **./client**
  - » **Enter the message to the server**

    xxxxxxx
- **Server : target**
  - » **The message from client :**

    xxxxxxx
- **Client : host**
  - » **The confirmation from the server**

Socket Server

```
#define PORTNUM       0x5005
#define BUFFSIZE 256

int main()
{
   int sockfd, newsockfd, clilen;
   char buffer[BUFFSIZE];
   struct sockaddr_in serv_addr, cli_addr;

   if((sockfd = socket(AF_INET, SOCK_STREAM, 0))== -1) {
     perror("ERROR opening socket");
     exit(1);
   }

   bzero((char *) &serv_addr, sizeof(serv_addr)); // write zeros to a byte string (man bzero)

   serv_addr.sin_family = AF_INET;
   serv_addr.sin_addr.s_addr = INADDR_ANY;
   serv_addr.sin_port = htons(PORTNUM); // from host byte order to network byte order (man
     htons)

   if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr))== -1){
     error("ERROR on binding");
     exit(1);
   }

   listen(sockfd,5);
```


```
clilen = sizeof(cli_addr);

    if((newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen))== -1){ #connect()
      from client
       error("ERROR on accept");
       exit(1);
    }

    bzero(buffer, BUFFSIZE);

    if((read(newsockfd,buffer, BUFFSIZE))== -1){
       error("ERROR reading from socket");
       exit(1);
    }
    printf("The message from client:₩n %s₩n",buffer);

    if((write(newsockfd,"The confirmation from server",28))== -1){
       error("ERROR writing to socket");
       exit(1);
    }
    close(sockfd); close(newsockfd);
}
```

Socket Client

```
#define PORTNUM         0x5005
#define BUFFSIZE   256
#define SERVERIP   "192.168.1.128"

int main()
{
   int sockfd;
   struct sockaddr_in serv_addr;
   char buffer[BUFFSIZE];

   if((sockfd = socket(AF_INET, SOCK_STREAM, 0))== -1){
      error("ERROR opening socket");
      exit(1);
   }

   serv_addr.sin_family = AF_INET;
   serv_addr.sin_port = htons(PORTNUM);
   serv_addr.sin_addr.s_addr = inet_addr(SERVERIP); / / from number-and-dots notation into
                                                    binary data in network byte order (man inet_addr)

   if(connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr))== -1){
      error("ERROR connecting");
      exit(1);
   }
```


```
printf("Please enter the message: ");
   bzero(buffer, BUFFSIZE);
   fgets(buffer,BUFFSIZE,stdin);

   if(write(sockfd,buffer,strlen(buffer))== -1){
      error("ERROR writing to socket");
      exit(1);
   }

   bzero(buffer,BUFFSIZE);
   if(read(sockfd,buffer, BUFFSIZE)== -1){
      error("ERROR reading from socket");
      exit(1);
   }
   printf("%s₩n",buffer);
   close(sockfd);
}
```

Web Server

- **An html web page is static (unchanging)**
  - Text document sent from server to browser
- **CGI program creates dynamic information**
  - Program is executed upon demand
  - Generates fresh content for each request
- **CGI (Common Gateway Interface)**
  - Used to communicate between the Web and your programs
  - Provides a way to make your Web pages more interactive
  - Standard programming interface to Web servers
  - CGI is not a programming language. It is just a set of standards (protocol)
- **Forms are a way to collect data from a user for processing by the server via your CGI script.**

CGI

- **Developer creates an HTML page with a <FORM> tag on it**

```
<form action="led.cgi" method="POST"> . . .
<input type="submit" name="button"...
                                    </form>
```

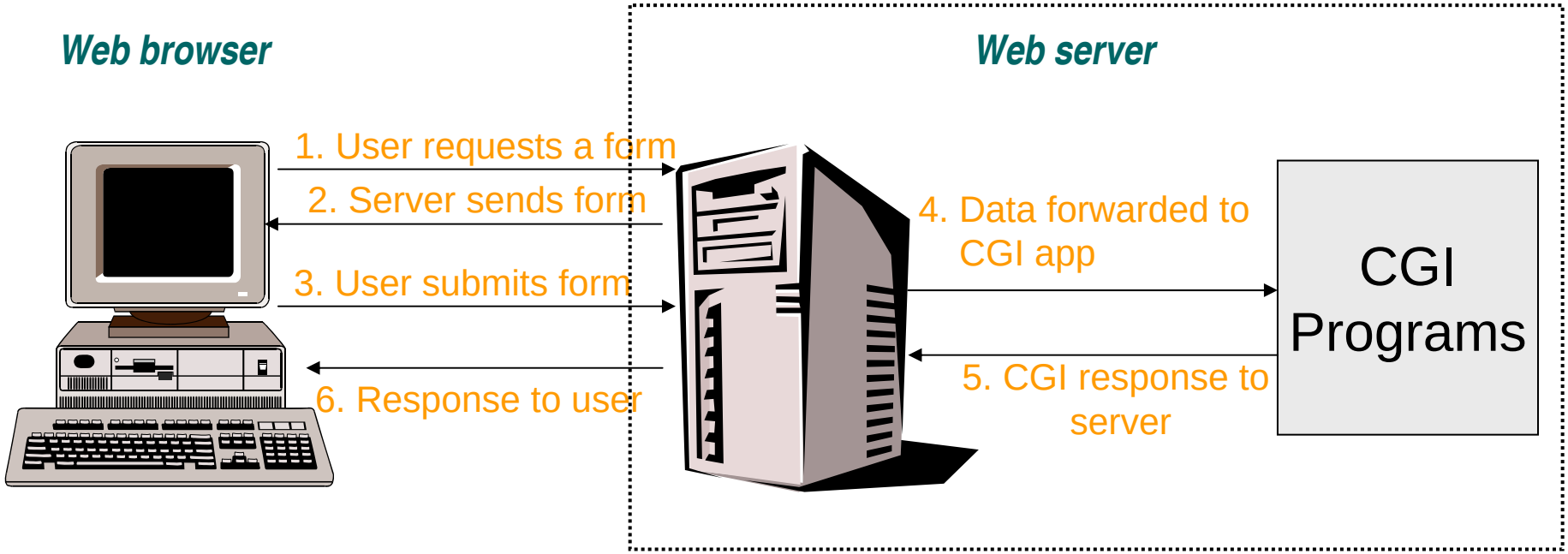
- **two ways to send data from a web form to a CGI program - method**
  - » **POST**
    - cgi program reads from stdin (the keyboard)
    - No limit on the amount of data sent
  - » **GET**
    - the input values from the form are sent as part of the URL and saved in the QUERY_STRING environment variable
    - cgi program reads from an environment variable (QUERY_STRING)
    - Limit on length of data sent

CGI

- User enters information into the Web page (fills in the variables in the <FORM> and clicks <SUBMIT>
  - The CGI program will be invoked every time the button SUBMIT is clicked.
- Forms receive input from
  - Text, Radio, Checkbox, Submit, ……
- The input, for a GET method, is sent as a line of data with the name=value pairs separated by &.
  - name=value&

CGI

- 1. The web browser requests a form from the server
- 2. The server send a form to the user.
- 3. The user fills out the form and "presses" the submit button
  - The browser sends the form's data to the server
- 4. The server recognizes the CGI call and passes the script name and the data (environmental variables) to the CGI program
- 5. The CGI application runs, usually generating a response to the server
- 6. The server passes the response back to the browser
  - The browser displays the response to the user


Web browser
Web server
1. User requests a form
2. Server sends form
3. User submits form
4. Data forwarded to CGI app
5. CGI response to server
6. Response to user
CGI Programs

Web Server

- **Web Server Source : goahead**
  - webserver.goahead.com
  - microcom.kut.ac.kr
- **Web Server Install**
  - tar xzvf web216.tar.gz
  - cd ws030325/linux
  - vi Makefile

```
CC=/opt/iwmmxt-1.0.0/bin/arm-linux-gcc
AR=/opt/iwmmxt-1.0.0/bin/arm-linux-ar
 :
$(CC) -c -o $@ $(DEBUG) $(CFLAGS) $(IFLAGS) $<
```

  - vi main.c

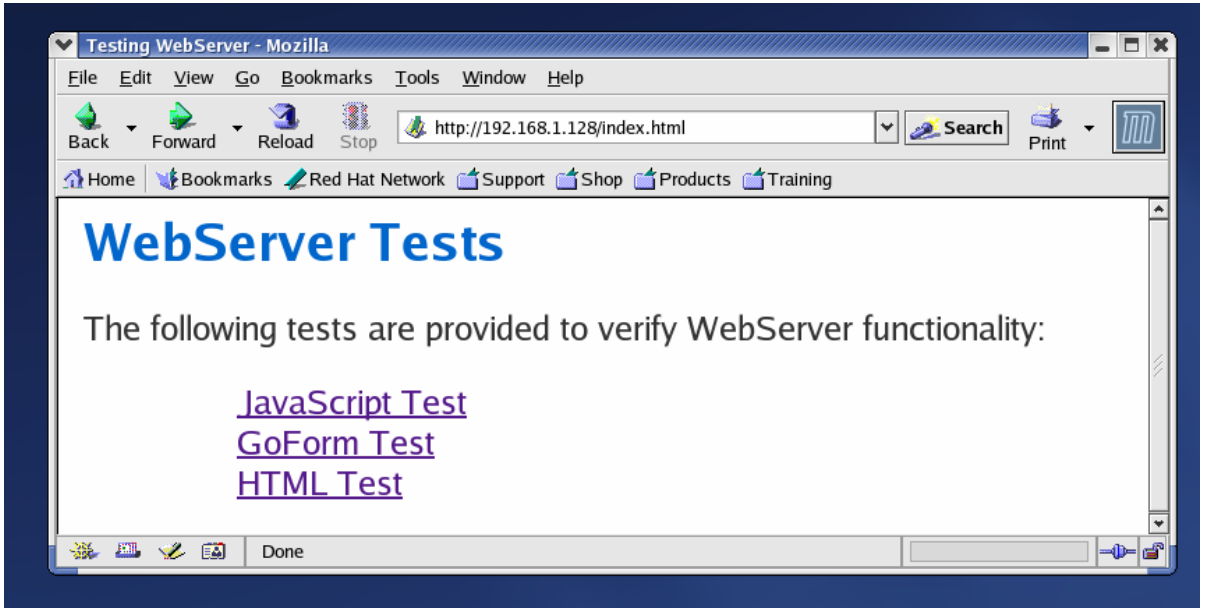
```
if (*url == '\0' || gstrcmp(url, T("/")) == 0) {
                        websRedirect(wp, T("index.html"));
                        return 1;
}
```

Web Server

- **Web Server Make : webs**
  - make
  - ls -al : webs
- **Web Test Files : web directory**
  - cd ws030325
  - tar cvf web.tar web
- **webs & web.tar : downloaded to Target**
  - cp web.tar / , and tar xvf web.tar
  - cp webs /bin
- **Target configuration**
  - vi /etc/hosts : 192.168.1.128 xhyper
  - ifconfig eth0 192.168.1.128 up
  - cd /web
  - cp tests.htm index.html

Web Server

- **Web Server Run : target**
  - **webs &**
- **Web Brower : host**
  - **Host Network Configuration**
    - “System Settings” -> “Network” -> “DNS” : local DNS
    - ifconfig eth0 192.168.1.100 up : local IP
  - **http://192.168.1.128**



CGI Test

- **LED CGI Test**
  - cp led-test.html /web target
  - cp led.cgi /web/cgi-bin
  - http://192.168.1.128/led-test.html host
  - Input 2 Hex Value
- **FND CGI Test**
  - cp fnd-test.html /web target
  - cp fnd.cgi /web/cgi-bin
  - http://192.168.1.128/fnd-test.html host
  - Input 2 Hex Value
- **CLCD CGI Test**
  - cp clcd-test.html /web target
  - cp clcd.cgi /web/cgi-bin
  - http://192.168.1.128/clcd-test.html host
  - Input Text String

led-cgi.c → led.cgi

```
int main()
{     char *cl;
      unsigned char val;
     unsigned int i;
   entry entries;

   printf("Content-type: text/html₩n₩n"); //CGI Program

   cl = (char *)getenv("QUERY_STRING");  //Get value from the user
   getinput_data(entries.name,cl,'=');
   getinput_data(entries.val,cl,'&');

   val = (unsigned char)strtol(entries.val,NULL,16); //string to long integer (hex)

   if(val == 0){
        if(!((entries.val[0] == '0'&& entries.val[1] == '₩0') ||
        (entries.val[0] == '0'&& entries.val[1] == '0')))
        {
                printf("<p>You entered the wrong value!");
                return 0;
         }
    }
  lightLed(val);
  printf("<br><center> * LED LIGHT ON/OFF * </center><br><hr>");

return 0;
}
```

led-cgi.c

```
#define LED_PHYS      0x12400000

typedef struct {
char name[128];
char val[128];
}entry;

void getinput_data(char *data, char *env_var, char stop) //get real input data from QUERY_STRING
{
      int x=0, y=0;

      for(x=0;((env_var[x]) && (env_var[x] != stop)); x++) data[x] = env_var[x];

      data[x] = '₩0';
      if(env_var[x]) ++x;

      while(env_var[y++] = env_var[x++]);
}
```

led-cgi.c

```
void lightLed(unsigned char val)
{
   int fd;
   int i;
   unsigned char *addr_led;

   if((fd=open("/dev/mem",O_RDWR|O_SYNC)) < 0) {
    printf("LED open fail₩n");
     exit(1);
    }

    addr_led = mmap(NULL,1,PROT_WRITE,MAP_SHARED,fd,LED_PHYS);
    *addr_led=val;
    munmap(addr_led, 1);
    close(fd);
}
```

led-test.html

```
<html>

<head>
<meta http-equiv="content-type" content="text/html; chatset=euc-kr">
<title>led cgi program</title>
</head>

<body bgcolor="#CCCCCC" text="black" link="blue" vlink="purple" alink="red">
<table align="center" cellpadding="0" cellspacing="0" width="250" height="80">
<tr>
     <td width="250" height="25" align="center"
valign="middle"></td>
</tr>
<tr>
  <td width="250" height="25" align="center" valign="middle">
<font size="2"> Input 2 Hex Value </font></td></tr>
<tr>
     <td width="250" align="center" valign="middle" height="30">
     <form method=get action="cgi-bin/led.cgi">
<p align="center">  0x <input type="text" name="value"
maxlength="2" size="2">
<input type="submit" name="button" value="input"></p></form>  </td>
</tr>
</table></body>

</html>
```