

Xilinx Zynq FPGA, TI DSP, MCU
기반의 프로그래밍 및 회로 설계
전문가 과정

<리눅스 네트워크 프로그래밍>
2018.04.03 - 29 일차

강사 - 이상훈
gcccompil3r@gmail.com

학생 - 안상재
sangjae2015@naver.com

1) gethostbyaddr() 함수를 이용해서 IP 주소(binary 형태)를 도메인 이름으로 바꿈.

- struct hostent* gethostbyaddr(const char *addr, int len, int type) : 네트워크 어드레스(IP 주소)에 대응하는 호스트 정보를 반환함.

- unsigned long inet_addr(const char *string) : Dotted-Decimal Notation 을 Big-Endian 32 비트 값(네트워크 바이트 순서)으로 변환

- char *inet_ntoa(struct in_addr addr) : 네트워크 바이트 순서의 32 비트 값을 Dotted-Decimal Notation 의 주소값으로 변환

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h>

typedef struct sockaddr_in si;

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc("\n", stderr);
    exit(1);
}

int main(int argc, char **argv)
{
    int i;
    si addr;
    struct hostent *host;

    if(argc != 2)
    {
        printf("use : %s <port>\n", argv[0]);
        exit(1);
    }

    memset(&addr, 0, sizeof(addr)); // addr 구조체 초기화
    addr.sin_addr.s_addr = inet_addr(argv[1]); /* 1 번째 인자(인터넷 점 표기법)를 네트워크 바이트 순서의 32
                                                비트 값으로 변환함 */
    host = gethostbyaddr((char *)&addr.sin_addr, 4, AF_INET); /* addr.sin_addr 주소에 대응하는 호스트 정보
                                                                를 반환함 */

    if(!host)
        err_handler("gethost error!");

    printf("Office Name : %s\n", host->h_name); // 도메인 이름 출력

    for(i=0; host->h_aliases[i]; i++)
        printf("Aliases %d : %s\n", i+1, host->h_aliases[i]); // 접속가능한 또 다른 도메인 이름(문자열)의 배열

    printf("Address Type : %s\n", (host->h_addrtype == AF_INET) ? "AF_INET" : "AF_INET6"); // 주소유형 출력

    for(i=0; host->h_addr_list[i]; i++)
        printf("IP Addr %d : %s\n", i+1, inet_ntoa(*(struct in_addr *)host->h_addr_list[i]));
        // 접속 가능한 또 다른 IP 주소의 배열

    return 0;
}
```

1-1) 결과 분석

- 127.0.0.1 local 주소를 입력했을 때 결과

```
sangjaeahn@sangjaeahn-900X5N:~/code/linux/network/4.3$ ./a.out 127.0.0.1
Office Name : localhost
Address Type : AF_INET
IP Addr 1 : 127.0.0.1
```

2. 메아리 프로그램

- 클라이언트 프로세스의 자식 프로세스에서 write 한 데이터를 서버 프로세스에서 read 하고 다시 write 하면 클라이언트의 부모 프로세스가 read 해서 화면에 출력함.

- 클라이언트 프로세스에서 fork()를 통해 프로세스를 생성하고, 부모 프로세스에서는 read 를 하고, 자식 프로세스에서는 write 를 수행함. 프로세스를 2 개로 나누어서 write, read 를 병렬적으로 수행함.

<서버 프로세스>

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <sys/socket.h>

typedef struct sockaddr_in si;
typedef struct sockaddr * sap;

#define BUF_SIZE 32

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

void read_childproc(int sig) // 자식 프로세스가 종료되었을 때 수행
{
    pid_t pid;
    int status;
    pid = waitpid(-1, &status, WNOHANG); // 자식 프로세스 처리
    printf("Removed proc id : %d\n", pid);
}

int main(int argc, char **argv)
{
    int serv_sock, clnt_sock;
    si serv_addr, clnt_addr;
    pid_t pid;
    struct sigaction act;
    socklen_t addr_size;
    int str_len, state;
    char buf[BUF_SIZE] = {0};

    if(argc != 2)
    {
        printf("use : %s <port>\n", argv[0]);
        exit(1);
    }
```

```
act.sa_handler = read_childproc; // signal 을 받았을 때 이동하는 함수
sigemptyset(&act.sa_mask); // act.sa_mask 를 비워줌
act.sa_flags = 0;
state = sigaction(SIGCHLD, &act, 0); // 자식 프로세스가 종료되면 act.sa_handler 함수로 이동

serv_sock = socket(PF_INET, SOCK_STREAM, 0); // 소켓파일을 만들고 fd 를 반환받음

if(serv_sock == -1)
    err_handler("socket() error");

memset(&serv_addr, 0, sizeof(serv_addr)); // serv_addr 구조체의 멤버들을 0 으로 초기화 함
serv_addr.sin_family = AF_INET; // 주소체계를 IPv4 로 설정함
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY); // 소켓이 동작하는 컴퓨터의 Ip 주소가 자동으로 할당됨
serv_addr.sin_port = htons(atoi(argv[1])); /* 첫번째 인자(포트번호)를 int 형으로 바꾸고 네트워크 바이트 순
                                             서로 바꾸어줌 */

if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr)) == -1) /* serv_sock 파일에 serv_addr 구조
                                                             체의 멤버들을 등록함 */
    err_handler("bind() error");

if(listen(serv_sock, 5) == -1) // 접속 가능한 클라이언트의 갯수를 5 개로 설정함
    err_handler("listen() error");

for(;;)
{
    addr_size = sizeof(clnt_addr);
    clnt_sock = accept(serv_sock, (sap)&clnt_addr, &addr_size);

    if(clnt_sock == -1)
        continue;
    else
        puts("New Client Connected ...");

    pid = fork();
    if(pid == -1)
    {
        close(clnt_sock);
        continue;
    }
    if(pid == 0) // 자식 프로세스에서 read, write 모두 수행
    {
        close(serv_sock);
        while((str_len = read(clnt_sock, buf, BUF_SIZE)) != 0) // read 는 blocking
            write(clnt_sock, buf, str_len); // 소켓파일로부터 read 한 데이터를 바로 write 함

        close(clnt_sock);
        puts("Client Disconnected ... ");
        return 0;
    }
    else
        close(clnt_sock);
}
close(serv_sock);
return 0;
}
```

////////////////////////////////////

<클라이언트 프로세스>

```
#include <stdio.h>
#include <stdlib.h>
```

```

#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

typedef struct sockaddr_in si;
typedef struct sockaddr * sap;

#define BUF_SIZE 32

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

void read_routine(int sock, char *buf)
{
    for(;;)
    {
        int str_len = read(sock, buf, BUF_SIZE);

        if(str_len == 0)
            return;

        buf[str_len] = 0;          // 문자열의 마지막을 널문자로 함
        printf("msg from server: %s", buf);
    }
}

void write_routine(int sock, char *buf)
{
    for(;;)
    {
        fgets(buf, BUF_SIZE, stdin); // 표준입력장치로부터 입력을 받음 ,read(0,buf,BUF_SIZE) 로도 구현 가능

        if(!strcmp(buf, "q\n") || !strcmp(buf, "Q\n"))
        {
            shutdown(sock, SHUT_WR); // 'q' 또는 'Q' 입력 시 자식 프로세스의 write 기능을 금지시킴
            return ;
        }
        write(sock, buf, strlen(buf)); // 소켓 파일에다가 표준입력장치로부터 받은 데이터를 write
    }
}

int main(int argc, char **argv)
{
    pid_t pid;
    int i, sock;
    si serv_addr;
    char buf[BUF_SIZE] = {0};

    if(argc != 3)
    {
        printf("use : %s <IP> <PORT>\n", argv[0]);
        exit(1);
    }

    sock = socket(PF_INET, SOCK_STREAM, 0);

    if(sock == -1)

```

```

        err_handler("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]); /* 1 번째 인자인 십진수 점표준 표기법 주소를 네트워크 이진형태
                                                    의 주소로 바꾸어줌 */

    serv_addr.sin_port = htons(atoi(argv[2]));

    if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1) // 서버와의 연결을 시도함
        err_handler("connect() error");
    else
        puts("Connected .....");

    pid = fork();

    if(pid == 0)
        write_routine(sock, buf); // 자식 프로세스는 write 만 함
    else
        read_routine(sock, buf); // 부모 프로세스는 read 만 함
    close(sock);
    return 0;
}

```

2-1) 결과 분석

- 클라이언트에서 데이터를 write 하면 서버에서 데이터를 read 해서 다시 write 해줌. 클라이언트에서 다시 read 해서 터미널 창에 출력함.

```

Connected .....
sdf
msg from server: sdf
asf
msg from server: asf
wef
msg from server: wef
w
msg from server: w
josephahn
msg from server: josephahn
sangjaeahn
msg from server: sangjaeahn
hi
msg from server: hi
hi
msg from server: hi
hihi
msg from server: hihi
hello
msg from server: hello

```

3) 채팅 프로그램

- 접속 가능한 클라이언트의 갯수를 10 개로 셋팅.
- 각각의 클라이언트에서 서버로 전송된 데이터는 서버에서 모든 클라이언트에게 브로드캐스트 함.

<서버 프로세스>

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/epoll.h>

#define BUF_SIZE 128
#define MAX_CLNT 256

typedef struct sockaddr_in si;
typedef struct sockaddr * sp;

int clnt_cnt = 0;
int clnt_socks[MAX_CLNT];
pthread_mutex_t mtx;

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc("\n", stderr);
    exit(1);
}

void send_msg(char *msg, int len)
{
    int i;

    pthread_mutex_lock(&mtx);

    for(i=0;i<clnt_cnt;i++)
        write(clnt_socks[i], msg, len); // 현재 접속한 모든 클라이언트의 소켓파일에 msg 를 write 함

    pthread_mutex_unlock(&mtx);
}

void *clnt_handler(void *arg) // 스레드 실행
{
    int clnt_sock = *((int *)arg);
    int str_len = 0,i;
    char msg[BUF_SIZE];

    while((str_len = read(clnt_sock, msg, sizeof(msg))) != 0) // 클라이언트로부터 전송된 데이터를 read 함
        send_msg(msg, str_len);

    pthread_mutex_lock(&mtx);

    for(i=0;i<clnt_cnt;i++) // 새로운 클라이언트가 접속되면 기존의 클라이언트와 중복이 되는지 체크함
    {
        if(clnt_sock == clnt_socks[i])
        {
            while(i++ < clnt_cnt - 1)
                clnt_socks[i] = clnt_socks[i+1]; // 중복되는 fd 는 삭제함
            break;
        }
    }
}
```

```

    }

    clnt_cnt--;
    pthread_mutex_unlock(&mtx);
    close(clnt_sock);
    return NULL;
}

int main(int argc, char **argv)
{
    int serv_sock, clnt_sock;
    struct serv_addr, clnt_addr;
    socklen_t addr_size;
    pthread_t t_id;

    if(argc != 2)
    {
        printf("Usage : %s <port>\n", argv[0]);
        exit(0);
    }

    pthread_mutex_init(&mtx, NULL);
    serv_sock = socket(PF_INET, SOCK_STREAM, 0);

    if(serv_sock == -1)
        err_handler("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(atoi(argv[1]));

    if(bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("bind() error!");

    if(listen(serv_sock, 10) == -1)
        err_handler("listen() error!");

    for(;;)
    {
        addr_size = sizeof(clnt_addr);
        clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_addr, &addr_size);

        pthread_mutex_lock(&mtx);
        clnt_socks[clnt_cnt++] = clnt_sock;    /* 접속된 클라이언트 갯수만큼 clnt_cnt에 저장하고, 각각의 fd
                                                를 clnt_socks 배열에 저장함 */
        pthread_mutex_unlock(&mtx);

        pthread_create(&t_id, NULL, clnt_handler, (void *)&clnt_sock); // 스레드 생성(clnt_handler 함수)
        pthread_detach(t_id); /* t_id 서브 스레드를 메인 스레드로부터 분리시킴(메인 스레드가 종료되어도
                                t_id는 무관함) */
        printf("Connect Client IP : %s\n", inet_ntoa(clnt_addr.sin_addr));
    }

    close(serv_sock);
    return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
<클라이언트 프로세스>
#include <stdio.h>

```



```

#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/epoll.h>

#define BUF_SIZE 128
#define NAME_SIZE 32

typedef struct sockaddr_in si;
typedef struct sockaddr * sp;

char name[NAME_SIZE] = "[DEFAULT]";
char msg[BUF_SIZE];

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

void *send_msg(void *arg)
{
    int sock = *((int *)arg);
    char name_msg[NAME_SIZE + BUF_SIZE];

    for(;;)
    {
        fgets(msg, BUF_SIZE, stdin); // 표준입력장치로부터 입력을 받아서 msg 에 저장함

        if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n"))
        {
            close(sock);
            exit(0);
        }

        sprintf(name_msg, "%s %s", name, msg); // name 과 msg 를 합쳐서 name_msg 배열에 저장함
        write(sock, name_msg, strlen(name_msg)); // 소켓파일에 name_msg 를 write 함
    }

    return NULL;
}

void *recv_msg(void *arg)
{
    int sock = *((int *)arg);
    char name_msg[NAME_SIZE + BUF_SIZE];
    int str_len;

    for(;;)
    {
        str_len = read(sock, name_msg, NAME_SIZE + BUF_SIZE - 1); /* 서버 프로세스에서 브로드 캐스트하는 데이터를 read 함*/

        if(str_len == -1)
            return (void *)-1;

        name_msg[str_len] = 0;
        fputs(name_msg, stdout); // 표준출력장치에 read 한 데이터를 출력함
    }
}

```

```

        return NULL;
    }

int main(int argc, char **argv)
{
    int sock;
    struct serv_addr;
    pthread_t snd_thread, rcv_thread;
    void *thread_ret;

    if(argc != 4)
    {
        printf("Usage : %s <IP> <port> <name>\n", argv[0]);
        exit(1);
    }

    sprintf(name, "[%s]", argv[3]); // 클라이언트 프로세스를 실행시킬 때 3 번째 인자를 닉네임으로 만들
    sock = socket(PF_INET, SOCK_STREAM, 0);

    if(sock == -1)
        err_handler("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
    serv_addr.sin_port = htons(atoi(argv[2]));

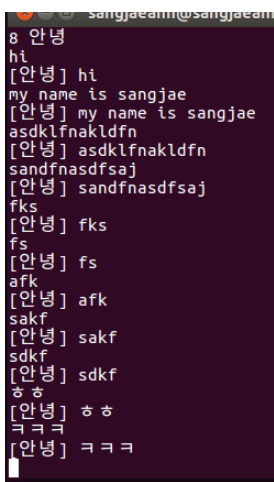
    if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) == -1) // 서버 프로세스에 연결함
        err_handler("connect() error!");

    pthread_create(&snd_thread, NULL, send_msg, (void *)&sock); // snd_thread 스레드 생성
    pthread_create(&rcv_thread, NULL, recv_msg, (void *)&sock); // rcv_thread 스레드 생성
    pthread_join(snd_thread, &thread_ret); // snd_thread 서버 스레드가 종료 될때까지 메인 스레드는 기다림
    pthread_join(rcv_thread, &thread_ret); // rcv_thread 서버 스레드가 종료 될때까지 메인 스레드는 기다림
    close(sock);
    return 0;
}

```

3-1) 결과 분석

- 클라이언트의 터미널창에 채팅 모습이 출력됨.



```

sangjae@sangjae:~$ ./chat_client 127.0.0.1 8080 sangjae
8 안녕
hi
[안녕] hi
my name is sangjae
[안녕] my name is sangjae
asdklfnakldfn
[안녕] asdklfnakldfn
sandfnasdfsaj
[안녕] sandfnasdfsaj
fks
[안녕] fks
fs
[안녕] fs
afk
[안녕] afk
sakf
[안녕] sakf
sdkf
[안녕] sdkf
ㅎㅎ
[안녕] ㅎㅎ
ㅋㅋㅋ
[안녕] ㅋㅋㅋ

```

4) 디버깅시 특정 코드부분의 수행시간을 출력함.

```
#include "load_test.h"

void get_runtime(tv start, tv end)
{
    end.tv_usec = end.tv_usec - start.tv_usec; // 수행시간의 마이크로초 단위를 구함
    end.tv_sec = end.tv_sec - start.tv_sec;     // 수행시간의 초 단위를 구함
    end.tv_usec += end.tv_sec * 1000000;

    printf("runtime = %lf sec\n", end.tv_usec / 1000000.0); // 수행시간을 마이크로단위로 출력함
}

#ifdef DEBUG // 디버깅 모드에만 실행하고, 일반모드에서는 실행하지 않음
int main(void)
{
    unsigned int i, cnt = 0;
    tv start, end;

    gettimeofday(&start, NULL); // 1970.01.01 로 부터 현재까지 시간을 마이크로초 단위까지 나타내는 함수

    for(i = 0; i < 777777777; i++)
        cnt++;

    gettimeofday(&end, NULL);

    get_runtime(start, end);

    return 0;
}
#endif
```

4-1) 결과 분석

- cpu 성능에 따라 runtime 시간은 달라질 수 있음

```
runtime = 1.537379 sec
```