TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-03-30 (27 회차)

강사: Innova Lee(이상훈)
gcccompil3r@gmail.com
학생: 정유경
ucong@naver.com

〈 네트워크바이트순서(big endian) ↔ Dotted-Decimal Notation 간의 상호변환 함수 〉

1. inet_aton() System Call - inet_addr.c

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

typedef struct sockaddr_in si;

void err_handler(char*msg)
{
    write(2, msg,strlen(msg));
    exit(1);
}

int main(int argc, char **argv)
{
    char *addr = "127.124.73.31";
        si addr_inet;

    if(!inet_aton(addr, &addr_inet.sin_addr))
        err_handler("Conversion Error!");
    else
        printf("Network Ordered Integer Addr: %x₩n", addr_inet.sin_addr.s_addr);

    return 0;
}
```

```
yukyoung@yukyoung-Z20NH-AS51B1U:~/Workspace/0330$ vi inet_aton.c
yukyoung@yukyoung-Z20NH-AS51B1U:~/Workspace/0330$ gcc inet_aton.c
yukyoung@yukyoung-Z20NH-AS51B1U:~/Workspace/0330$ vi inet_aton.c
yukyoung@yukyoung-Z20NH-AS51B1U:~/Workspace/0330$ ./a.out
Network Ordered Integer Addr: 1f497c7f
yukyoung@yukyoung-Z20NH-AS51B1U:~/Workspace/0330$
```

*. 지난시간에 했던 inet_addr(argv[1])은 IP 주소 192.168.1.102(10 진수, 점표현)를 네트워크 주소로 바꾼다. 즉, 빅엔디안 32 비트(16 진수 4 바이트, 66 01 a8 c0)로 변경해주었다.
inet_aton(argv[1], &addr_inet.sin_addr)은 inet_addr(argv[1])의 업그레이드 버전이다.
이 함수도 IP 주소를 네트워크 주소로 바꾸어준다.

*. 네트워크 주소는 '빅엔디안'으로 크로스매칭 시킨다.
(네트워크 주소는 빅엔디안으로 정해져 있어 기준이 된다)

호스트(로컬)		네트워크		호스트(로컬)
리틀	→	빅	→	빅
빅	→	빅	→	리틀

*. 127 . 124 . 73 . 31 을 네트워크 주소로 바꾸면 크로스 매칭되어 1f 49 7c 7f 가 된다.
(31→1f , 64+9→49, 128-4→7c, 128-1→ 7f)

2. inet_ntoa System Call - inet_ntoa.c

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

typedef struct sockaddr_in si;
```

```
int main(int argc, char **argv)
{
	si addr1, addr2;
	char *str;
	char str_arr[32]={0};

	addr1.sin_addr.s_addr=htonl(0x10203040); // hton( ) : 네트워크바이트 순서(빅엔디안)로 변경
	addr2.sin_addr.s_addr=htonl(0x12345678);

	str = inet_ntoa(addr1.sin_addr); // inet_ntoa( ) : 16 진수, 32 비트 네트워크주소를 10 진수, 점표기로 바꾼다
	strcpy(str_arr,str);
	printf("Not 1: %s₩n",str);

	inet_ntoa(addr2.sin_addr); // 컴파일러가 자동으로 str 에 값 저장
	printf("Not 2: %s₩n", str);
	printf("Not 3: %s₩n", str_arr);

	return 0;
}
```

```
yukyoung@yukyoung-Z20NH-AS51B1U:~/Workspace/0330$ vi inet_ntoa.c
yukyoung@yukyoung-Z20NH-AS51B1U:~/Workspace/0330$ gcc inet_ntoa.c
yukyoung@yukyoung-Z20NH-AS51B1U:~/Workspace/0330$ ./a.out
Not 1: 16.32.48.64
Not 2: 18.52.86.120
Not 3: 16.32.48.64
yukyoung@yukyoung-Z20NH-AS51B1U:~/Workspace/0330$
```

*. 호스트 → (aton) → 네트워크 → (ntoa) → 호스트
*. inet_ntoa(add1.sin_addr) 은 32 비트 네트워크 주소(16 진수)를 IP 주소(10 진수, 점표기)로 바꾼다
hton(0x10203040) → 16.32.48.64
// 리틀엔디안 타입의 0x102030 은 htonl 로 빅엔디안으로 크로스매칭 후 inet_ntoa()에 의해 10 진수, 점표기로 바뀐다.

hton(0x12345678) → 18.52.86.120

*. inet_ntoa()함수의 리턴값 문자열의 저장소는 함수 내부에 선언되어있는 static 버퍼이다.
따라서 addr2 의 주소정보를 가지고 inet_ntoa()함수를 호출하게 되면 이 버퍼의 값은 re-write 된다.
따라서 str 포인터를 한번 초기화 하고 같은 값을 두번 출력했을때 다른 값이 출력되는 것을 알 수 있다.
str = **inet_ntoa(addr1.sin_addr);**
inet_ntoa(addr2.sin_addr);
변경된 문자열 정보를 계속 유지하려면 따로 복사해서 값을 보관해야한다.
inet_ntoa(addr2.sin_addr);

3. Echo_Server, Echo_Client - echo_server.c , echo_client.c

gcc -o serv echo_server.c
gcc -c clnt echo_client.c
./serv 7777
./clnt <ip> 7777 (ip: 127.0.0.1)

```
/* echo_server.c */
#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 1024

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


int main(int argc, char **argv)
{
	int i, str_len;
	int serv_sock, clnt_sock;

	char msg[BUF_SIZE];

	si serv_addr, clnt_addr;
	socklen_t clnt_addr_size;

	if(argc != 2)
	{
		printf("use: %s <port>₩n", argv[0]);
		exit(1);
	}
	serv_sock = socket(PF_INET, SOCK_STREAM, 0); // 소켓생성(fd 반환)

	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);
/*INADDR_ANY : 이식성, 다수의 IP 주소에 대한 연결요청 처리 */
	serv_addr.sin_port = htons(atoi(argv[1])); // 문자열→ 숫자→ 빅엔디안

	if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr))== -1) // IP 주소, PORT 번호 할당
	err_handler("bind() error");

	// isten()은 논블록킹 accept()는 블록킹 - connect()되어야 값을 반환함
	if(listen(serv_sock, 5) == -1) // 6 명부터 서비스? ?  // 연결요청대기
	err_handler("listen() error");

	clnt_addr_size = sizeof(clnt_addr);

	for(i=0; i<5;i++) // ARM 은 1byte 연산 지원안함, 4byte(int 형) 처리에 최적화 되어있다.
	{	// accept( ) : 클라이언트 전용소켓 생성
		clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr, &clnt_addr_size); // 연결승
인
		if(clnt_sock == -1)
			err_handler("accept() error");
		else
		printf("Connected Client %d₩n", i+1);
		// read( ) : blocking
		while((str_len = read(clnt_sock, msg, BUF_SIZE)) != 0) // client socket 에서 읽어서 msg 에
저장
		write(clnt_sock, msg, str_len); // msg 를 client socket 에 쓴다 <echo>
		close(clnt_sock); // 클라이언트용 소켓을 닫고, 다른 클라이언트의 접속을 기다린다.
	}
	close(serv_sock);
```

```
	return 0;

}
```

```
/*echo_client.c*/
/* 실행: gcc -o serv echo_server.c
gcc -c clnt echo_client.c
./serv 7777
./clnt <ip>
*/

#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 1024
void err_handler(char * msg)
{
	fputs(msg, stderr);
	fputc('₩n', stderr);
	exit(1);
}

int main(int argc, char **argv)
{
	int sock, str_len;
	si serv_addr;
	char msg[32];
	char *m = "Input Message(q to quit) ";

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

	sock = socket(PF_INET, SOCK_STREAM,0); // 소켓생성 (fd 반환)

	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]));

// connect(0) : sock 파일에 서버의 IP 주소와 PORT 번호 등록 → 서버에서 accept( )
	if(connect(sock, (sap) &serv_addr, sizeof(serv_addr))==-1)
		err_handler("connect() error");
	else
	puts("Connected.....");

	for(;;)
	{
		fputs("Input msg(q to quit): ",stdout); // 표준출력으로 " ~" 출력
		fgets(msg, BUF_SIZE,stdin); // 표준입력으로 BUF_SIZE 만큼 msg[32]에 입력받는다.

		if(!strcmp(msg,"q₩n") || !strcmp(msg,"Q₩n"))
```

```
                break; // 루프탈출조건

                write(sock, msg, strlen(msg)); // msg[32]를 sock 에 쓴다.(서버에 보낸다)
                str_len = read(sock, msg, BUF_SIZE - 1); // sock 에서 읽어온다(서버에서 받는다) 〈echo〉

                if(str_len == -1)
                        err_handler("read() error!");
                msg[str_len] = 0; // 마지막에 널문자
                printf("msg from serv: %s₩n", msg); // msg[32] 출력
        }
        close(sock);
        return 0;
}
```

*. for 반복문 구조에서 char 형을 사용하지 않고 메모리 낭비되게 int 형 변수 i 를 사용하는 이유
ARM Architecture 의 구조가 4byte (int 형)처리에 최적화 되어있기 때문이다.
(ARM 은 1byte 연산 지원안함, CISC 구조인 intel 만 4byte 를 안써도 성능저하가 없다.)

*. INADDR_ANY
1. 서버의 IP 주소를 자동을 찾아서 대입, 다른서버컴퓨터에 프로그램이 설치되어도 IP 주소를 변경할 필요 없음
(즉, 소스수정하지 않아서 이식성이 좋다)
2. 한 컴퓨터에 LAN 카드(NIC)가 여러개일 경우 특정 NIC 의 IP 주소를 지정하지 않고 INADDR_ANY 를 써서 여러 IP 주소를 통해 들어오는 모든 연결 요청을 받아서 처리할 수 있다.

*. 결과: 클라이언트 쪽에 〈echo〉만 되다가 close( )하면 서버측에서 read( ) 즉, blocking 함수가 blocking 하고 있던 값이 쭉 출력된다.
→ 수정방법: 1. read, write 를 논 블록킹으로 바꾼다, 2. server 에서 client 를 accept() 할 때마다 fork( ) 하여 각 client 의 메시지를 개별적으로 read( )하여 블록??

4. op_client, op_server

```
/*op_client.c*/

#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        1024
#define RLT_SIZE        4 // 결과가 정수형이므로 4
#define OPSZ    4

void err_handler(char*msg)
{
        fputs(msg, stderr); // stdout , stderr 나눠서 출력하는 이유? stderr 출력하면서 프로세스 종료??
        fputc('₩n', stderr);
        exit(1);
}
```

```
int main(int argc, char **argv)
{
	int i, sock, result, opnd_cnt;
	char opmsg[BUF_SIZE] ={0};
	si serv_addr;

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

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


	fputs("Operand Cnt: ", stdout); // 몇개의 인자를 연산할 것인가
	scanf("%d", &opnd_cnt);

	opmsg[0] = (char)opnd_cnt;

	for(i=0;i<opnd_cnt;i++)
	{
		printf("Operand %d:", i+1); // 1 번부터 받는다
		scanf("%d", (int*) &opmsg[i*OPSZ+1]);
	}

	fgetc(stdin); // 표준입력 'c'
	fputs("Operator: ", stdout); // 표준출력 "~"
	scanf("%c", &opmsg[opnd_cnt * OPSZ+ 1]); // 연산자 입력
	write(sock, opmsg, opnd_cnt*OPSZ+2); // 서버에 전송, 입력받은 연산자까지 (+2)???
	read(sock, &result, RLT_SIZE);

	printf("Operation result: %d₩n", result);
	close(sock);
	return 0;
}
```

```
/* op_server.c */

#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	1024
```

```
#define OPSZ   4 // 정수연산이라서 4 (연산하는 인자 자체의 크기)

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

int calculate(int opnum, int *opnds, char op)
{

	int result = opnds[0]; // 결과를 먼저저장? ? ?
	int i;

	switch(op)
	{
		case'+':
			for(i=1; i<opnum; i++) // 0 은 result 로 사용중 (1 ~ opnum-1)
				result += opnds[i];
			break;

		case'-':
			for(i=1; i<opnum; i++)
				result -= opnds[i];
			break;

		case'*':
			for(i=1; i<opnum; i++)
				result *= opnds[i];
			break;

	}
	return result;

}


int main(int argc, char **argv)
{

	int serv_sock, clnt_sock;
	char opinfo[BUF_SIZE];

	int result, opnd_cnt, i;
	int recv_cnt, recv_len;

	si serv_addr, clnt_addr;
	socklen_t clnt_addr_size;

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

	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, (sap) &serv_addr, sizeof(serv_addr))==-1)
		err_handler("bind() error");

	if(listen(serv_sock, 5)== -1)
		err_handler("listen() error");

	clnt_addr_size = sizeof(clnt_addr);

	for(i=0;i<5;i++)
	{
		opnd_cnt = 0;
		clnt_sock = accept(serv_sock, (sap) &clnt_addr, &clnt_addr_size);
		read(clnt_sock, &opnd_cnt,1); // client 소켓에서 읽어서 int opnd_cnt 에 저장

		recv_len=0;
		while((opnd_cnt * OPSZ +1) > recv_len)
		{
			recv_cnt = read(clnt_sock, &opinfo[recv_len], BUF_SIZE-1); // 읽은만큼만 버퍼에
쓴다
			recv_len += recv_cnt; // 읽은 바이트수를 +
		}

		result = calculate(opnd_cnt, (int*)opinfo, opinfo[recv_len-1]);
/*
opnd_cnt : 몇개의 인자를 연산할 것인가 (3 입력시 3 개)
char opinfo[BUF_SIZE] , opinfo[recv_len-1]  → 'operator' (12 바이트 읽으면 실제 배열에는 0~11 까지 들어간
다.) */
		write(clnt_sock, (char*) &result, sizeof(result));
// int result 를 (char*)로 형변환? ? 클라이언트도 int 형인데?

		close(clnt_sock);
	}
	close(serv_sock);
	return 0;
}
```