TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-04-02 (28 회차)

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

1. 숫자맞추기(UP&DOWN) 게임 - gclint, gserver.c

```
/*gclinet*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>  /* -lpthread 붙여서 컴파일*/
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/epoll.h>

#define BUF_SIZE                128

typedef struct sockaddr_in      si;
typedef struct sockaddr *       sp;

char msg[BUF_SIZE];

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

/* fork()와 동일, 송수신분리위해 클라이언트에서 송신용, 수신용 클라이언트 생성*/
void *send_msg(void *arg)  // void * arg = &thread_ret = int sock
{                          // 어떤 자료형, 구조체가 들어올 지 모르므로 void 형으로 만듬
        // 구조체로 묶어서 자료형이 다른 여러개 인자를 전달인자로 전달할 수도 있음
       int sock = *((int *)arg);  // 서버소켓 int sock
        char msg[BUF_SIZE];
        for(;;) // 무한루프 돌면서 사용자가 입력한 내용을 계속 서버로 전송한다.
        {
                fgets(msg, BUF_SIZE, stdin); // 표준입력으로 msg 받는다
                write(sock, msg, strlen(msg)); // 서버에 msg 전송
        }
        return NULL;
}

void *recv_msg(void *arg)
{
        int sock = *((int *)arg);
        char msg[BUF_SIZE];
        int str_len;
        for(;;) // 무한루프 돌면서
        {
                str_len = read(sock, msg, BUF_SIZE - 1); // 서버에서 msg 로 수신
                msg[str_len] = 0; // 0 으로 잘라준다 → 0 뒤의 부분만 읽는다? ?
                fputs(msg, stdout); // fouts stdout 은 write 0 과 동일
        }
        return NULL;
}

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

        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, (sp)&serv_addr, sizeof(serv_addr)) == -1)
                err_handler("connect() error");

        pthread_create(&snd_thread, NULL, send_msg, (void *)&sock); // thread 동작전 (생성만 한 상태)
        pthread_create(&rcv_thread, NULL, recv_msg, (void *)&sock);
        pthread_join(snd_thread, &thread_ret);// join 순간 thread 가 동작을 시작한다
        pthread_join(rcv_thread, &thread_ret);

        close(sock);

        return 0;
}
```

```
/*gserv.c*/

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

#define BUF_SIZE        128
#define  MAX_CLNT       256 // listnen 2 명인데 256 은 어디서 나온 숫자일까? ?

typedef struct sockaddr_in              si;
typedef struct sockaddr *               sp;

int clnt_cnt = 0;
int clnt_socks[MAX_CLNT];
int data[MAX_CLNT];
int thread_pid[MAX_CLNT];
int idx;
int cnt[MAX_CLNT];
pthread_mutex_t mtx; //&mtx 가 락의 키값이 된다

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

void sig_handler(int signo)
{
        int i;
        printf("Time Over!₩n");
        pthread_mutex_lock(&mtx); // 임계영역 starts
        for(i = 0; i < clnt_cnt; i++)
                if(thread_pid[i] == getpid()) // Threadpid == 현재 프로세스 id 이면
                        cnt[i] += 1; // 카운트 1 증가
        pthread_mutex_unlock(&mtx); // 임계영역 ends
        alarm(3); // 3 초 후 SIGALRM 발생
}
```

```
void proc_msg(char *msg, int len, int k) // (클라이언트가 입력한 숫자, str_len, 현재 인덱스 i)
{
	int i;
	int cmp = atoi(msg);
	char smsg[64] = {0};

	pthread_mutex_lock(&mtx);
	cnt[k] += 1; // 입력한 횟수 카운트
	if(data[k] > cmp)
		sprintf(smsg, "greater than %d₩n", cmp);
	else if(data[k] < cmp)
		sprintf(smsg, "less than %d₩n", cmp); // smsg 에 문자열 저장+0(자동)
	else
	{
		strcpy(smsg, "You win!₩n"); // smsg 에 문자열을 복사한다
		printf("cnt = %d₩n", cnt[k]);
	}
	strcat(smsg, "Input Number: ₩n"); // 문자열(string)에 이어붙인다(널문자부터)
	write(clnt_socks[k], smsg, strlen(smsg));

#if 0
	for(i = 0; i < clnt_cnt; i++)
	{
		if(data[i] > cmp)
			sprintf(smsg, "greater than %d₩n", cmp);
		else if(data[i] < cmp)
			sprintf(smsg, "less than %d₩n", cmp);
		else
			strcpy(smsg, "You win!₩n");

		strcat(smsg, "Input Number: ");
		write(clnt_socks[i], smsg, strlen(smsg));
	}
#endif

	pthread_mutex_unlock(&mtx);
}

void *clnt_handler(void *arg)
{
	int clnt_sock = *((int *)arg); // int 형 포인터로 형변환(캐스팅)한 arg 가 가리키는 값
	int str_len = 0, i;
	char msg[BUF_SIZE] = {0};
	char pattern[BUF_SIZE] = "Input Number: ₩n";

	signal(SIGALRM, sig_handler);

	pthread_mutex_lock(&mtx);
	thread_pid[idx++] = getpid(); // 쓰레드의 pid 저장 - 0 이 나오는게 맞나요?
	i = idx - 1; // 현재 인덱스 파악
	printf("i = %d₩n", i);
	write(clnt_socks[i], pattern, strlen(pattern));
	pthread_mutex_unlock(&mtx);

	alarm(3);
// while 로 맞출때까지 반복, 맞추고 나서도 반복된다. (str_len !=0 이므로)
	while( (str_len = read(clnt_sock, msg, sizeof(msg))) != 0) // 클라이언트가 입력한 숫자→ msg 로 수신
	{
		alarm(0); // 이전에 설정된 알람제거
		proc_msg(msg, str_len, i);
```

```
                alarm(3); // 3 초뒤 SIGALRM
        }
        pthread_mutex_lock(&mtx);

        for(i = 0; i < clnt_cnt; i++)
        {
                if(clnt_sock == clnt_socks[i])
                {
                        while(i++ < clnt_cnt - 1)
                                clnt_socks[i] = clnt_socks[i + 1];
                        break;
                }
        }

        clnt_cnt--;
        pthread_mutex_unlock(&mtx);
        close(clnt_sock);

        return NULL;
}

int main(int argc, char **argv)
{
        int serv_sock, clnt_sock;
        si serv_addr, clnt_addr;
        socklen_t addr_size;
        pthread_t t_id;
        int idx = 0;
        if(argc != 2)
        {
                printf("Usage: %s <port>₩n", argv[0]);
                exit(1);
        }
        srand(time(NULL));
        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, (sp)&serv_addr, sizeof(serv_addr)) == -1)
                err_handler("bind() error");
        if(listen(serv_sock, 2) == -1) // 2 명이상 접속불가
                err_handler("listen() error");
        for(;;)
        {
                addr_size = sizeof(clnt_addr);
                clnt_sock = accept(serv_sock, (sp)&clnt_addr, &addr_size);
// 무한루프 돌면서 accept, 다음 클라이언트 까지 블로킹, client 와 연결된 fd 반환
                thread_pid[idx++] = getpid(); // 왜하나요?  // 쓰레드의 pid 저장, getpid 값이 왜 0 이 나올까요??
// 소켓파일은 공유메모리라서 락을 걸어주어야 다른 쓰레드들이 접근하여 중구난방으로 값이 바뀌지 않음
                pthread_mutex_lock(&mtx);
                data[clnt_cnt] = rand() %10+ 1; // rand() % 3333+ 1;
                clnt_socks[clnt_cnt++] = clnt_sock;
                pthread_mutex_unlock(&mtx);

                pthread_create(&t_id, NULL, clnt_handler, (void *)&clnt_sock);
                pthread_detach(t_id); // 쓰레드 자원할당 해제 or 쓰레드와 프로세스를 분리하여 별도로 동작
                printf("Connected Client IP: %s₩n", inet_ntoa(clnt_addr.sin_addr)); // 십진수 점표기로 바꾼
```

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

*. int pthread_mutex_init(pthread_mutex_t * mutex, const pthread_mutex_attr *attr);
pthread_mutex_init 는 mutex 객체를 초기화
1st mutex 객체
2nd attr : mutex 특성 (기본 mutex 특성: NULL, fast, recurisev, error checking 의 종류가 있으며, 디폴트로 fast 가 사용)

*. int pthread_create(&snd_thread, NULL , send_msg , (void *) &sock); → pthread 를 생성한다
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);
1st pthread 식별 자
2nd pthread 옵션 (기본 = NULL)
3rd pthread 로 분기하여 실행할 함수 → void* handler (void* arg)
4th 분기할 함수[3]로 넘겨줄 인자값 (어떤 자료형을 넘겨줄지 모르기 때문에 void 형으로 넘겨주고 상황에 맞게 함수 내에서 원래의 자료형으로 캐스팅해서 사용)
- 0 반환값 0(성공)

*. pthread_join(snd_thread, &thread_ret);
→ 특정 pthread 가 종료될 때까지 기다리다가 종료시 자원해제
1st pthread 식별자
2nd pthread 의 return 값 (void** thread_return)

*. pthread_detach(t_id); → t_id 식별자를 가지는 pthread 가 부모 pthread 로부터 독립
(독립된 pthread 는 따로 pthread_join()이 없어도 종료시 자동으로 리소스 해제)
'일반적'으로 쓰레드를 pthread_create() 를 사용하여 생성하면, 쓰레드가 종료되더라도 사용했던 모든 자원이 해제되지 않음, 반면 pthread_join() 을 사용하여 종료될때까지 기다렸다가 종료시점이 되면, 자원이 반납. pthread_detach() 함수는 pthread_join()을 사용하지 않더라도, 쓰레드 종료될때 모든 자원을 해제.

*. sprintf (char * **str**, const char * format, ...);
1st 문자열이 저장될 char 배열을 가리키는 포인터
2nd str 에 쓰여질 문자열
printf 에서 화면에 출력하는 대신에 화면에 출력할 **문자열**을 str 에 쓴다는 것이다. 이 때, 인자로 지정한 배열의 크기는 배열에 쓰여질 문자열의 크기 보다 커야함. (주의) sprintf 함수는 자동적으로 str 맨 마지막에 NULL 문자를 붙임 한 칸의 여유 필요함
sprintf(buf,"character : %c ₩n", character);
printf("%s ₩n", buf);
→ (출력) character : c

0 을 1 로 바꾸면서 코드를 넣었다 뺐다 할때	두개의 코드를 번갈아 테스트 할때 (0,1 을 번갈아)	3 개 이상의 코드의 경우

#if 0
/* meaningless codes */
#endif
	#if 0
	
	/* excluded */
	#else
	/* included */
	#endif
		#define FLAG 2
		
		#if FLAG == 0
		printf("FLAG is 0");
		#endif
		#if FLAG == 1
		printf("FLAG is 1");
		#endif
		#if FLAG == 2
		printf("FLAG is 2");**#endif**

2. 파일전송 file_client, file_server.c

```
/*file_server*/

#include <fcntl.h>
#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);
}

int main(int argc, char*argv[])
{
        int serv_sock, clnt_sock, fd;
        char buf[BUF_SIZE] = {0};
        int read_cnt;

        si serv_addr, clnt_addr;
        socklen_t clnt_addr_size;

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

        fd = open("file_server.c",O_RDONLY);  // 클라이언트에 보내는 정보 // 다른 파일명 or argv[2] 도 가능
        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);
        clnt_sock = accept(serv_sock, (sap)&clnt_addr, &clnt_addr_size);

        for(;;)
        {
                read_cnt = read(fd, buf, BUF_SIZE);
                if(read_cnt < BUF_SIZE) // 읽은것이 buf 보다 작다면
                {
                        write(clnt_sock, buf, read_cnt); // 쓰고 종료
                        break;
                }
// if 읽을것이 더 있다면 for 문돌면서 read & write 한다.
```

```
//→ 다시 read 할때 어디서부터 read? fd 내의 offset 이 이동하나요?
                write(clnt_sock, buf, BUF_SIZE);
        }

        shutdown(clnt_sock, SHUT_WR); // clnt_sock 의 WR 만 닫고 RD 는 열어둔다
        read(clnt_sock, buf, BUF_SIZE);
        printf("msg from client: %s₩n", buf);

        close(fd);
        close(clnt_sock);
        close(serv_sock);

        return 0;
}
```

```
/*file_client.c*/

#include <fcntl.h>
#include <stdlib.h>
#include <stdio.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);
}

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

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

    fd=open("receive.txt", O_CREAT|O_WRONLY, 0644); // 권한문제로 파일이 열리지 않아서 권한설정함
    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.....");

    while((read_cnt = read(sock, buf, BUF_SIZE)) !=0 ) // 읽을 데이터가 있으면 계속 읽어서 buf 에 저장한다
    write(fd, buf, read_cnt);

    puts("Received File Data");
    write(sock, "Thank you",10); // 서버에 Thank you 를 전송 :)
    close(fd);
    close(sock);

    return 0;
}
```

*. Socket close()와 shutdown() 차이 비교

- 둘다 네트워크 연결을 종료시킨다,
- TCP 는 send buffer 와 recv buffer 가 있다.

[1] Close(socket 의 fd) : file descriptor 로 지정된 파일 서술자와, 그 파일 서술자가 가리키는 실제 파일과의 연결을 차단.

- socket()함수로 소켓을 열면 참조 카운터가 1 증가. 다른 자식 프로세스로 복사될 때도 참조 카운터가 1 증가.
- close 는 참조 카운터를 1 감소 → 참조카운터가 0 이 되면 소켓을 닫는다.

(닫힌 소켓은 더이상 사용할 수 없다. 소켓 참조 카운터가 0 이 아니라면 열린 상태)

(shutdown 은 참조카운터에 상관없이 연결을 종료함)

[2] Shutdown(int sockfd, int howto): 소켓 함수로 생성된 파일 서술자에는 (소켓 버퍼라고도 한다.) 송신 버퍼와 수신 버퍼 두가지가 열려있는데, shutdown()함수로 각각의 부분적 연결을 차단 가능.

- close()함수는 양방향(send recv) 둘 다 종료시키는데 반해, shutdown 함수는 howto 인자에 따라 동작이 다름.

→ close 함수의 약점: close()호출 후에 받을 데이터가 있다면 받을 수 없다.
(그러나 shutdown 의 howto 인자를 설정하면 가능하다)
SHUT_RD : 연결의 recv 한쪽만 닫는다. 이제 이 소켓으로는 데이터를 받을 수 없다.

수신버퍼도 폐기된다.

SHUT_WR : 연결의 send 한쪽만 닫는다. 이제 이 소켓으로는 어떤 데이터도 보낼 수 없다.
송신버퍼에 남아 있는 데이터는 모두 보낸 뒤에 TCP 연결을 종료한다.

→ 만약에 자신은 데이터를 다 보냈다 하면, SHUT_WR 인자를 설정하여 shutdown()을 호출하여
다른 쪽이 보내는 데이터를 받을 수 있게 한다.

3. gethostbyname()

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

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

int main(int argc, char* argv[])
{
    int i;
    struct hostent *host;

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

    host = gethostbyname(argv[1]);
    if(!host)
        err_handler("gethost.....error!");
    printf("Official Name: %s₩n", host->h_name);

    for(i=0;host->h_aliases[i];i++)
        printf("Aliases %d: %s₩n", i+1, host->h_aliases[i]);

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

    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]));
    // in_addr 구조체 변수 포인터
    return 0;
}
```

*. sockaddr_in 이라는 구조체의 정보는 다음과 같다.

```
struct sockaddr_in {
  sa_family_t  sa_family;    // 주소 체계
  uint16_t     sin_port;     // 16 비트 TCP/UDP 포트
  struct in_addr   sin_addr;  // 32 비트 IPv4 주소
  struct in_addr {
    uint32_t    s_addr;
  }
  char sin_zero[8];          // 사용되지 않음
}
struct sockaddr_in {
  sa_family_t  sa_family;    // 주소 체계
  uint16_t     sin_port;     // 16 비트 TCP/UDP 포트
  struct in_addr   sin_addr;  // 32 비트 IPv4 주소
  struct in_addr {
    uint32_t    s_addr;
  }
  char sin_zero[8];          // 사용되지 않음
}
```

struct sockaddr_in.sin_addr.s_addr 에는 32 비트 IPv4 주소를 넣을 수 있다.

*. IP 주소를 출력할 때에는 inet_ntoa() 함수를 이용하여 네트워크 바이트 순서의 32 비트 값을 점이 찍힌 십진수 표현방식으로 바꾸어 출력한다.

*. gethostbyname() : 도메인 이름으로 hostent 정보(IP 주소 등등)를 구함
〈netdb.h〉 struct hostent *gethostbyname(const char *name);

주어진 호스트 name 에 상응하는 **hostent 타입의 구조체**를 반환한다. hostent 구조는 아래와 같습니다.

```
struct hostent
{
  char *h_name;             /* Official name of host.  공식 도메인 이름*/
  char **h_aliases;         /* Alias list.  공식 이름 외의 별칭 */
  int h_addrtype;           /* Host address type. IP 주소체계  */
  int h_length;            /* Length of address. 결과로 전달될 IP 주소의 길이 */
  char **h_addr_list;       /* List of addresses from name server.  */
// 도메인 이름에 해당하는 IP 주소들 여러개의 IP 가 배열로 저장*/

#define h_addr  h_addr_list[0]  /* Address, for backward compatibility.  */
};
```

cf. gethostbyaddr() 함수를 이용하여 리턴값을 hostent 타입으로 선언된 구조체에 넣을 수 있다.
gethostbyaddr() 함수는 IP 주소를 이용하여 도메인 네임을 얻으려고 할 때 보통 사용할 수 있다.