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

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

네트워크

술게임만들기

술게임

서버

```
/* For Network */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>

/* For System */
#include <time.h>
#include <fcntl.h>
#include <signal.h>
#include <stdbool.h>
#include <sys/wait.h>

typedef struct sockaddr_in      si;
typedef struct sockaddr *       sap;

#define BUF_SIZE                1024
#define OPSZ                    4

int glob_cnt;

void sig_handler(int signo)
{
        printf("Time Over\n");
        glob_cnt++;
        alarm(3);
}
```

클라이언트

```
#include <stdio.h>
#include <fcntl.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
#define OPSZ                    4

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

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

        if(argc != 3)
        {
```

```
void make_game(int *data)
{
	*data = rand() % 3333 + 1;
}

bool check_correct(int data, int cmp)
{
	if(data == cmp)
		return true;

		return false;
}
void start_game(int data, int sock) //반복
{
	char buf[32] = {0};
	char msg[128] = {0};
	char out[64] = "숫자를 맞춰봐: ";
	bool fin;
	int i, cmp, nr;

	signal(SIGALRM, sig_handler);

	for(;;)
	{
		write(sock, out, strlen(out));
		alarm(3);
		//read(0, buf, sizeof(buf));
		nr = read(sock, buf, sizeof(buf));// 블록킹
		buf[nr - 1] = '\0';
		alarm(0);
		cmp = atoi(buf); // 받은 문자열 정수로

		fin = check_correct(data, cmp); // 판별
```

```
		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 .....");
	for(;;)
	{
		nread = read(sock, buf, BUF_SIZE);
		//buf 에 값저장, nread 에 길이 저장
		buf[nread] = '\0';
		write(1, buf, nread); // printf
		nread = read(0, buf, sizeof(buf));//블록킹
		buf[nread] = '\0';
		write(sock, buf, nread);
		// 성공 및 종료 조건 코드 추가
		nread = read(sock, buf, BUF_SIZE);
		buf[nread] = '\0'; // 마지막값에
		write(1, buf, nread);
	}

	close(sock);
```

```
            if(fin)  //같다면
            {
                  break;
            }
            else
            {
                  glob_cnt++;
                  if(data > cmp)
                  {
                        sprintf(msg, "%d 보다 크다\n", cmp);// msg 값쓰기.
                        msg[strlen(msg)] = '\0';
                        write(sock, msg, strlen(msg)); //clnt 에 보내깅
                        //printf("%d 보다 크다\n", cmp);
                  }
                  else
                  {
                        sprintf(msg, "%d 보다 작다\n", cmp);
                        msg[strlen(msg)] = '\0';
                        write(sock, msg, strlen(msg));
                        //printf("%d 보다 작다\n", cmp);
                  }
            }
      }
}

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

```
      return 0;
}
```

```
void child_proc(int sig)
{
	pid_t pid;
	int status;
	pid = waitpid(-1, &status, WNOHANG);
	printf("Removed proc id: %d\n", pid);
}

int main(int argc, char **argv)
{
	//pid_t pid[5] = {0};
	pid_t pid;
	int status;

	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;

	struct sigaction act;
	int state;

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

	act.sa_handler = child_proc;
	sigemptyset(&act.sa_mask);
```

```
	act.sa_flags = 0;
	state = sigaction(SIGCHLD, &act, 0);

	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");

	for(i = 0; i < 5; i++)  // 5 명 들어올 수 있어.
	{
		clnt_addr_size = sizeof(clnt_addr);
		clnt_sock = accept(serv_sock, (sap)&clnt_addr,
&clnt_addr_size);

		pid = fork(); //fork 로 프로세스 복사

		if(pid == 0)  // 부모 wait 아니었나.. 우선 ㄱ
		{
			int data;

			srand(time(NULL));

			close(serv_sock); // 서버 소켓 닫는다.
```

```
                make_game(&data);  // 랜덤
                printf("data = %d\n", data); // 데이터확인.
                start_game(data, clnt_sock); // 데이터, 클라이언트 소켓

               printf("Client Disconnected\n");
               close(clnt_sock);
              }
    close(serv_sock);
    return 0;
}
```

→ 실행 터미널 두개로하면 잘안돼다. 이유는?

→ fork 를 왜할까?

같은일을 여러개가 동시에 처리하기위해서 server 에서 다수의 client 로 부터 요청이 들어오면 동시처리를 해야함으로 수신부분에서 업무가 있을 때마다 복제하여 각각의 클라이언트로 전송한다.

82 서버 예제

```
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdlib.h>
#include <stdio.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]);     // 인터넷 서버
   if(!host)
      err_handler("gethost ... error!");

   printf("Official Name: %s\n", host→h_name); //host 구조체 이름 출력, 서버 구조체 char[] 배열로 내보낼수있음.
```

```
    for(i=0;host->h_aliases[i]; i++)
        printf("alisases %d: %s\n", i+1, host->h_aliases[i]); //aliases?? 이차원포인터배열

    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]));

    return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/network/4_1$ ./a.out naver.com
Official Name: naver.com
Address Type: AF_INET
IP Addr 1: 125.209.222.142
IP Addr 2: 210.89.160.88
IP Addr 3: 210.89.164.90
IP Addr 4: 125.209.222.141
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/lhs/network/4_1$ ./a.out daum.net
Official Name: daum.net
Address Type: AF_INET
IP Addr 1: 211.231.99.80
IP Addr 2: 203.133.167.81
IP Addr 3: 203.133.167.16
IP Addr 4: 211.231.99.17
```

→ 리눅스용 소켓프프로그래밍: 도메인이름을 이용해서 ip 주소 알아내기 .

*목요일 까지 숙제

만약 시간이 남는다면 369 게임을 만들어 보자! 박수=ctrl+c , SIGALRM 은 2 초

복습중…

18 일차 코드는 이렇게 짜자..

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include "my_scanf.h"

int main(void)
{
        int nr;
        char buf[1024]={0};

        nr= my_scanf(buf,sizeof(buf));
        printf("nr=%d\n",nr);
        write(1,buf,nr);
        return 0;
}

```

```c
#include "my_scanf.h"

int my_scanf(char *buf, int size)
{
        int nr = read(0,buf,size);
        return nr;
}
```

write(1,~) : 모니터에 쓰기

```c
#ifndef __MY_SCANF_H__ // 선언된 것이 있으면 이 헤더파일을 들어가지마라.
#define __MY_SCANF_H__ // 선언된 것이  없으면 my_scanf_h 가 0 으로 됨. (그냥상수임!)

#include <fcntl.h>
#include <unistd.h>

int my_scanf(char *, int);

#endif    // 두번 선언하는 것을 방지한다.


// 분할돼어있어 분간이쉬움
```

* stderr (standard error)으로 에러 메세지를 화면으로 출력하는이유 printf 로 하면 돼지 않나??

출력이 복잡할 경우에 stderr 의 효과를 볼 수 있다. printf 를 쓸 경우 버퍼 문제로 제대로 출력이 되지 않을 경우가 있다. 이경우에 에러메시지출력을 제때, 제대로 못하게 되는 경우가 생긴다. Stderr 같은 경우는 버퍼 없이 바로 출력한다. 따라서 어떤 상황이 와도 가장 빠르게 에러 메세지를 출력 할 수가 있다. (파일 포인터)

stdin: 표준입력장치, 보통은 키보드

stdout: 표준출력장치, 보통은 모니터

stderr: 표준에러출력장치, 보통은 모니터

Fprintf → printf 는 무조건 화면으로 출력, fprintf 함수는 스트림을 지정해줄 수 있는 함수이다. fprintf(stderr,…)

19 일차 → 파일 묶기 , 풀기 이해 노

* t 로 끝나는 자료형의 정체

ssize_t = signed int

size_t = unsigned int

off_t = long형 정수

시스템의 종속적 요소로서 int 나 기타 포인터의 크기는 그 시스템의 워드(시스템이 한번에 읽어내는 메모리의 크기)와 같아서 32bit 시스템에서는 4byte, 16bit시스템에서는 2byte 시스템에 따라 달라지기 때문에 시스템이 변경되면 코드를 수정해야만 한다.

그러나 _t 로 끝나는 자료형을 사용하면 시스템이 변경하게 되어도 다른 적절한 자료형을 가지고 헤더파일에 선언되어 있는 자료형들을 재정의해주기만 하면 되기 때문이다. <sys/types.h> 안에 들어 있다.

즉, 특정 자료형에 종속되지 않도록 새로운 이름의 자료형을 정의해주는 것이 가능해진다.

-->파일보고있음..
1 pthread 활용법

2 네트웍 플밍 기본기
3. 기타정리