TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/4/2
수업일수	28 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

-up,down game

-file 전송

-gethostbyname

-pthread 활용법

1. up down game 해석

-server.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

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;

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

```
void sig_handler(int signo)
{
	int i;

	printf("Time Over!₩n");

	pthread_mutex_lock(&mtx); //임계영역 잠금

	for(i = 0; i < clnt_cnt; i++)
		if(thread_pid[i] == getpid())//pid 값과 thread_pid 가 같으면
			cnt[i] += 1;//cnt 배열에 1을 더함
		//왜 비교해서 배열에 저장하는건지 모르겠음
	pthread_mutex_unlock(&mtx); // 임계영역 잠금 해제

	alarm(3);//알람 3초
}

void proc_msg(char *msg, int len, int k)
{
	int i;
	int cmp = atoi(msg);//입력한 값을 숫자값으로 변환
	char smsg[64] = {0};

	pthread_mutex_lock(&mtx);

	cnt[k] += 1;//몇번 입력했는지

	if(data[k] > cmp)//입력값이 작으면
		sprintf(smsg, "greater than %d₩n", cmp);
	else if(data[k] < cmp)//입력값이 크면
		sprintf(smsg, "less than %d₩n", cmp);
```

```
		else//같으면
		{
			strcpy(smsg, "You win!₩n");
			printf("cnt = %d₩n", cnt[k]);
		}

		strcat(smsg, "Input Number: ₩n");
		write(clnt_socks[k], smsg, strlen(smsg));//클라이언트 소켓에 정보 전달

#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);//unlock 해줌
}

void *clnt_handler(void *arg)//arg 로 pthread 마지막 인자가 들어옴 fd 값 들어옴
{
		int clnt_sock = *((int *)arg);
		int str_len = 0, i;
```

```
	char msg[BUF_SIZE] = {0};
	char pattern[BUF_SIZE] = "Input Number: ₩n";

	signal(SIGALRM, sig_handler);//3초내에 입력안하면 sig_handler 실행

	pthread_mutex_lock(&mtx);//lock 해줘서 critical section 을 막아줌
	thread_pid[idx++] = getpid();//쓰레드의 pid 값을 저장
	i = idx - 1;//현재 인덱스 값
	printf("i = %d₩n", i);
	write(clnt_socks[i], pattern, strlen(pattern));
	//clnt_socks 첫번째 클라이언트에 pattern 을 써줌
	pthread_mutex_unlock(&mtx);//unlock 해줌, 쓰레드가 여러개 될 수 있으니깐(클라이언트가
여러명이면)

	alarm(3);//3초 대기

	while((str_len = read(clnt_sock, msg, sizeof(msg))) != 0)
	{			//클라이언트가 숫자 입력한걸 읽음
		alarm(0);//알람꺼줌
		proc_msg(msg, str_len, i);
		alarm(3);//proc_msg 가 끝났으니 알람 3해줘서 맞출 때 까지 반복
	}

	pthread_mutex_lock(&mtx);//임계영역 잠금

	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)//인자가 2개가 아닐 경우 오류메세지 출력
                {
                        printf("Usage: %s <port>₩n", argv[0]);
                        exit(1);
                }

                srand(time(NULL));//랜덤출력을 위한 설정

                pthread_mutex_init(&mtx, NULL);//mtx 를 초기에 NULL 로 초기화

                serv_sock = socket(PF_INET, SOCK_STREAM, 0);//서버의 소켓 파일디스크립터 저장
```

```
	if(serv_sock == -1)//소켓 에러
		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)//서버소켓에 주소 부여
		err_handler("bind() error");

	if(listen(serv_sock, 2) == -1)//클라이언트를 2명 받음
		err_handler("listen() error");

	for(;;)//무한루프
	{
		addr_size = sizeof(clnt_addr);
		clnt_sock = accept(serv_sock, (sp)&clnt_addr, &addr_size);
		//서버 프로그램이 클라이언트의 연결 요청을 받고 서버소켓과는 다른 소켓을 생성하여
파일 디스크립터를 반환받음.
		//클라이언트의 커넥트를 승인함
		//다음 클라이언트가 올 때 까지 블록킹
		thread_pid[idx++] = getpid(); //배열에 프로세스 아이디를 저장

		pthread_mutex_lock(&mtx);//lock 하는 이유 : 데이터가 꼬이지 말라고 걸어줌
					//socket 파일이 서버랑 클라이언트랑 공유되어서
lock 해줘야함
		data[clnt_cnt] = rand() % 3333 + 1;//1~3333까지의 랜덤값이 data 배열에 저장
		clnt_socks[clnt_cnt++] = clnt_sock;//clnt_socks 배열에 클라이언트 소켓 저장
		pthread_mutex_unlock(&mtx);//lock 을 풀어줌
```

```
                pthread_create(&t_id, NULL, clnt_handler, (void *)&clnt_sock);
                        //t_id 쓰레드 아이디 값 생성,  ,쓰레드가 되는 함수, 쓰레드에 전달되는 인자
                pthread_detach(t_id);//t_id 를 분리시킨다

                printf("Connected Client IP: %s₩n", inet_ntoa(clnt_addr.sin_addr));
        }

        close(serv_sock);

        return 0;
}
```

```
-client.c
#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

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);
}

void *send_msg(void *arg)
{
		int sock = *((int *)arg);
		char msg[BUF_SIZE];

		for(;;)//계속 사용자의 입력을받고 전송하는걸 반복
		{
			fgets(msg, BUF_SIZE, stdin);//입력을 받아 msg 에 저장
			write(sock, msg, strlen(msg));//sock 에 write 해서 서버로 전송
		}

		return NULL;
}

void *recv_msg(void *arg)
{
		int sock = *((int *)arg);//int 형 포인터로 형변환 한 arg 의 포인터를 입력
		char msg[BUF_SIZE];
		int str_len;

		for(;;)
		{
			str_len = read(sock, msg, BUF_SIZE - 1);//sock 을 읽어 msg 에 넣어줌,
			//write 0으로 해줘도 된다 ??
			msg[str_len] = 0;//맨마지막에 0을 해줌으로서 이전의 길었던 뒤의 값을 덮어쓰지 않도록
해줌
```

```
                fputs(msg, stdout);
        }

        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);//소켓생성

        if(sock == -1)//소켓에러
                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)//소켓하고 커넥트 한 순가 서버의
accept 동작
                err_handler("connect() error");

        pthread_create(&snd_thread, NULL, send_msg, (void *)&sock);//송신
                                                        //4번째 인자 값은 (void*)&sock : 쓰레드가
구동시키려는 함수의 인자값
```

```
                                                        //void 포인터로 보내는게 어떤인자가 올지
모르니까
        pthread_create(&rcv_thread, NULL, recv_msg, (void *)&sock);//수신
                                                        //(void*)&sock : 구동시키려는 함수의
인자값
        //송신과 수신을 분리하기 위해 thread 만듦 (fork 랑 같음)
        //create 는 만드는것 join 은 동작하는것
        pthread_join(snd_thread, &thread_ret);
        pthread_join(rcv_thread, &thread_ret);
        //,,두 쓰레드가 죽었을 때 close 함
        close(sock);

        return 0;
}
```

2.file 전송

-file_server.c

```
#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);
serv_sock = socket(PF_INET, SOCK_STREAM, 0); //소켓생성

if(serv_sock == -1)//소켓에러
   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)//bind 에러
    err_handler("bind() error");

if(listen(serv_sock, 5) == -1) //클라이언트를 5명 받음
   err_handler("listen() error");

clnt_addr_size = sizeof(clnt_addr);//clnt_addr 사이즈를 받음

clnt_sock = accept(serv_sock, (sap)&clnt_addr, &clnt_addr_size);//클라이언트의 요청을 받음

for(;;)
{
    read_cnt = read(fd, buf, BUF_SIZE);//file_server.c 를 읽어 buf 에 저장
```

```
        if(read_cnt < BUF_SIZE)//32보다 file_server.c 의 byte 수가 작을 경우
        {
            write(clnt_sock, buf, read_cnt);//clnt_sock 에 buf 를 쓴다
            break;//for 문 빠져나감
        }
            write(clnt_sock, buf, BUF_SIZE);
    }

    shutdown(clnt_sock, SHUT_WR);//소켓을 닫음
    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 <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)
{
    char buf[BUF_SIZE] = {0};
    int fd, sock, read_cnt;
    si   serv_addr;

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

  fd = open("receive.txt", O_CREAT | O_WRONLY);//파일 디스크립터 반환
  sock = socket(PF_INET, SOCK_STREAM, 0);//소켓생성

  if(sock == -1)//소켓에러
     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)//서버와 connect
      err_handler("connect() error");
  else//성공하면 출력
      puts("Connected.....");

  while((read_cnt = read(sock, buf, BUF_SIZE)) != 0)//sock 을 읽어 buf 에 저장
      write(fd, buf, read_cnt);//fd 에 buf 를 쓴다

  puts("Receibed File Data");
  write(sock, "Thank you", 10);
  close(fd);
  close(sock);

  return 0;
}
```

~결과

```
xeno@xeno-NH:~/Downloads$ ./clnt 127.0.0.1 7777
Connected.....
Receibed File Data
```

receive.txt 파일이 전송된다.

3.gethostbyname.c

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

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

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

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

	host = gethostbyname(argv[1]);//host name 을리턴

	if(!host)//host 가 값이 없으면 에러메세지 출력
		err_handler("gethost ... error!");

	printf("Official name : %s₩n",host→h_name);//오피셜 네임 출력

	for(i=0; host->h_aliases[i];i++)//별칭을 출력. 없을 수 도 있다.
		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");//IPv4이면 AF_INET 을 IPv6이면 AF_INET6 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]));//ip 주소 값들을 출력

        return 0;
}
```

```
xeno@xeno-NH:~/proj/0402$ ./a.out naver.com
Official name : naver.com
Address type : AR_INET
IP Addr 1 : 210.89.164.90
IP Addr 2 : 125.209.222.142
IP Addr 3 : 125.209.222.141
IP Addr 4 : 210.89.160.88
xeno@xeno-NH:~/proj/0402$ ./a.out daum.net
Official name : daum.net
Address type : AR_INET
IP Addr 1 : 203.133.167.81
IP Addr 2 : 211.231.99.80
IP Addr 3 : 211.231.99.17
IP Addr 4 : 203.133.167.16
xeno@xeno-NH:~/proj/0402$ ./a.out google.com
Official name : google.com
Address type : AR_INET
IP Addr 1 : 216.58.199.110
```

-AF_INET , PF_INET

둘 다 상수 값을 가지긴 하나 프로토콜 정보를 설정하는 부분에는 PF_INET 을 주소 정보를 설정하는 부분은 AF_INET 을 설정하는 것이 좋음.

AF_INET : 주소 체계 중 하나

PF_INET : 프로토콜 체계(프로토콜 패밀리) 중 하나

4.pthread 활용법

pthread(POSIX thread)

<pthread.h> 헤더파일에 존재함.

컴파일 할 때 -lpthread 옵션을 주어야 한다.

pthread_t : pthread 의 자료형을 의미한다.

pthread_create : pthread 를 생성한다.

형태 : int pthread_create(pthread_t *thread, const pthread_attr_t *attr,void *(*start_routine) (void *), void *arg);

인자 :

*thread : 쓰레드가 성공적으로 생성되면 thread 식별 값이 주어진다.

*attr : pthread 속성, 기본적인 thread 속성을 사용 할 때에는 NULL

*(*start_routine) : pthread 로 분기 할 함수. 반환값이 void*,매개변수도 void*

*arg : 분기할 함수로 넘겨줄 인자 값. 어떤 자료형을 넘겨줄 지 모르기 때문에 void 로 넘겨주고 함수 내에서 원래의 자료형으로 형변환해서 사용한다.

리턴 : 성공적으로 pthread 생성되면 0반환, 실패시 에러번호

pthread_join : 특정 pthread 가 종료될 때 까지 기다리다가 특정 pthread 가 종료시 자원 해제시켜줌

형태 : int pthread_join(pthread_t thread, void **retval);

인자 :

thread : 어떤 pthread 를 기다리는지 정하는 식별자

**retval : pthread 의 반환 값. 포인터로 값을 받아온다.

ptrhead_detach : 쓰레드를 분리시킴. 일반적으로 pthread_create 를 사용하면 쓰레드가 종료되더라도 사용한 자원이 해제되지 않고 반면에 pthread_join 를 사용하면 종료될 때 자원이 해제된다. ptread_detach 는 pthread_create 를 종료될 때 모든 자원을 해제하게 해줌.

형태 : int pthread_detach(pthread_t thread);

인자

thread : 분리시킬 쓰레드 식별자

리턴 : 성공하면 0, 실패하면 에러코드 반환

pthread_exit

형태 : void pthread_exit(void *retval);

인자

*retval : 현재 실행중인 thread 를 종료시킬 때 사용

pthread_self : 현재 실행중인 pthread 의 식별자를 반환

형태 : pthread_t pthread_self(void);

pthread_cleanup_push : pthread_exit 가 호출될 때 호출된 handler 를 정하는 함수. mutex lock 를 해제할 때 사용

형태 :void pthread_cleanup_push(void (*routine)(void *), void *arg);

인자

pthread_cleanup_pop : cleanup handler 를 제거하기 위해 사용되는 함수

형태 : void pthread_cleanup_pop(int execute);

인자 : execute 가 0일 경우 바로 cleanup handler 를 제거하고 그 외의 값을 가질 때는 한번 실행한 후 제거한다.

Mutex 는 여러 개의 쓰레드가 공유하는 데이터를 보호하기 위해 사용되는 도구로서 한번에 하나의 쓰레드만 실행 가능하도록 하여 공유되는 데이터를 보호한다.

pthread_mutex_init : mutex 객체를 초기화 시키기 위해 사용한다.

형태 : int pthread_mutex_init(pthread_mutex_t * mutex, const pthread_mutex_attr *attr);

인자

mutex : 인자 mutex 를 초기화 시킴
attr : 특성 변경, 기본 특성 사용하려면 NULL 을 사용한다. fast, recurisev, error checking 이 있다.

pthread_mutex_t 구조체
pthread_mutex_lock : critical section 시작
pthread_mutex_unlock : critical section 종료