TI DSP,Xilinx zynq FPGA,MCU 및 Xilinx zynq FPGA 프로그래밍 전문가 과정

강사-INNOVA LEE(이상훈)
Gccompil3r@gmail.com
학생-윤지완
Yoonjw7894@naver.com

<clnt>
```
#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);//fputs() 함수는 string을 현재
위치의 출력 stream으로
복사합니다. 스트링 끝의 null 문자 (\0)를
복사하지 않습니다.fgetc() : 파일로부터
한 문자씩 데이터를 불러온다.
fputc('\n', stderr);//fputc() : 파일에 한 문자씩
데이터를 입력한다.
Exit(1);
}

void *send_msg(void *arg)
{
int sock = *((int *)arg);//서버 socket을
받아온다.
char msg[BUF_SIZE];
        for(;;)
{
fgets(msg, BUF_SIZE, stdin);//스트림에서
문자열을 받아서 (num - 1) 개의 문자를 입력
```

```
받을 때 까지나, 개행 문자나 파일
끝(End-of-File) 에 도달할 >때 까지 입력
받아서 C 형식의 문자열로 저장한다.
write(sock, msg, strlen(msg));
}

        return NULL;
}

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

       for(;;)
{
str_len = read(sock, msg, BUF_SIZE - 1);

                    msg[str_len] = 0;
fputs(msg,
stdout);//writ(0,msg,sizeof(str_len))으로 써도
무방하
다.
}

        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);
//송신과 수신을 분리하기 위해서 사용,이것은
fork와 같은 동작
//thread의 4번째는 thread함수의 동작하는 인자.
pthread_create(&rcv_thread, NULL, recv_msg,
(void *)&sock);
pthread_join(snd_thread, &thread_ret);//pthread가
본격적으로 동작을 하고
종료 될때는 ctrl+c를 눌렀을 때다.
pthread_join(rcv_thread, &thread_ret);

        close(sock);

        return 0;
}
```

<serv>

```
#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)
{
fputs(msg, stderr);
fputc('\n', stderr);
exit(1);
}

void sig_handler(int signo)
{
int i;

        printf("Time Over!\n");

        pthread_mutex_lock(&mtx);

        for(i = 0; i < clnt_cnt; i++)
if(thread_pid[i] == getpid())
cnt[i] += 1;

        pthread_mutex_unlock(&mtx);

        alarm(3);
}

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

        pthread_mutex_lock(&mtx);

        cnt[k] += 1;
```

```
        if(data[k] > cmp)
sprintf(smsg, "greater than %d\n", cmp);
else if(data[k] < cmp)
sprintf(smsg, "less than %d\n", cmp);
else
{
strcpy(smsg, "You win!\n");
printf("cnt = %d\n", cnt[k]);
}

        strcat(smsg, "Input Number: \n");
write(clnt_socks[k], smsg, strlen(smsg));
pthread_mutex_unlock(&mtx);
}

void *clnt_handler(void *arg)//마지막 인자가 들어온다
clnt_sock의 pid가 넘어온다
{
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);

        pthread_mutex_lock(&mtx);//lock을 건이유는
cretical section을 하기 위해>서다.
thread_pid[idx++] = getpid();//thread의 pid값을
저장한다.
i = idx - 1;
printf("i = %d\n", i);
write(clnt_socks[i], pattern, strlen(pattern));//첫번째
clnt
pthread_mutex_unlock(&mtx);

        alarm(3);

        while((str_len = read(clnt_sock, msg,
sizeof(msg))) != 0)
{
alarm(0);
proc_msg(msg, str_len, i);
alarm(3);
```

```
}
        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;//thread의 ip or 식별자라고도 한다.
int idx = 0;
        if(argc != 2)
{
printf("Usage: %s <port>\n", argv[0]);
exit(1);
}
        srand(time(NULL));
        pthread_mutex_init(&mtx, NULL);//처음에는
lock걸게 없으니 0으로 초기화한
다.
        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)
err_handler("listen() error");

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

                thread_pid[idx++] = getpid();

                pthread_mutex_lock(&mtx);//lock의
key값을 의미한다.데이터가 꼬이
지 않게 이 순간부터는 다른것들은 들어오지 말라는 뜻.
data[clnt_cnt] = rand() % 3333 + 1;
clnt_socks[clnt_cnt++] = clnt_sock;
pthread_mutex_unlock(&mtx);//작업이 다 끝나면 이제
lock을 풀고 >할거하라는 명령어.

                pthread_create(&t_id, NULL,
clnt_handler, (void *)&clnt_sock);
//(void *)&clnt_sock은 thread가 받을 인자.
pthread_detach(t_id);//프로세스랑 thread랑 떼어낸다
그럼 별도로

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

        close(serv_sock);

        return 0;
}
```

<IP확인 하는 예제>

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

host = gethostbyname(argv[1]);
if(!host)
err_handler("gethost ..... error!");
printf("office name :%s\n", host->h_name);
for(i=0;host->h_aliases[i]; i++)
printf("aleases%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,//naver.com을
하면 네이버 IP번호로 들어간다.
inet_ntoa(*(struct in_addr *)host->h_addr_list[i]));
return 0;
```

<thread 정리>

1.1. Thread vs Process

Thread는 프로세스와 다음과 같은 차이점을 가진다.

프로세스는 독립적이다. 쓰레드는 프로세스의 서브셋이다.
프로세스는 각각 독립적인 자원을 가진다. 쓰레드는 stat, memory 기타 다른 자원들을 공유한다.
프로세스는 자신만의 주소영역을 가진다. 쓰레드는 주소영역을 공유한다.
프로세스는 IPC(:12)를 이용해서만 통신이 가능하다.
일반적으로 쓰레드의 문맥교환(context switching)는 프로세스의 문맥교환보다 빠르다.

1.2쓰레드의 생성과 종료

멀티 쓰레드 프로그램이라고 하더라도, 처음 시작되었을 때는 main()에서 시작되는 단일 쓰레드 상태로 작동이 된다. 이 상태에서 pthread_create(3) 함수를 호출함으로써, 새로운 쓰레드를 생성할 수 있다. pthread_create를 이용해서 생성된 새로운 쓰레드를 worker 쓰레드라고 하자.

멀티 쓰레드 프로그램은 다음과 같은 흐름을 가진다.2

1.3 pthread_create : 쓰레드 생성

pthread_create(3)함수를 이용하면 새로운 쓰레드를 생성할 수 있다. 이 함수는 다음과 같이 사용할 수 있다.

thread : 쓰레드가 성공적으로 생성되었을 때, 넘겨주는 쓰레드 식별 번호.
attr : 쓰레드의 특성을 설정하기 위해서 사용한다. NULL(:12)일 경우 기본 특성
start_routine : 쓰레드가 수행할 함수로 함수포인터(:12)를 넘겨준다.
arg : 쓰레드 함수 start_routine를 실행시킬 때, 넘겨줄 인자

1.4 pthread_join : 쓰레드 정리

쓰레드가 실행시키는 것은 함수 이다. 그러므로 return이나 exit(0)등을 이용해서 쓰레드를 종료시킬 수 있게 된다. 그러나 쓰레드 함수가

종료되었다고 해서 곧바로 쓰레드의 모든자원이 종료되지 않는다. fork()기반의 멀티프로세스 프로그램에서 종료된 자식프로세스를 정리하기 위해서 wait()로 기다리듯이, 종료된 쓰레드를 기다려서 정리를 해주어야만 한다. 그렇지 않을 경우 쓰레드의 자원이 되돌려지지 않아서 메모리 누수현상이 발생하게 된다.

pthread_create()로 생성시킨 쓰레드는 pthread_join()을 통해서 기다리면 된다. pthread_join 함수는 다음과 같이 사용할 수 있다.

th : pthread_create에 의해서 생성된, 식별번호 **th**를 가진 쓰레드를 기다리겠다는 얘기다.
thread_return : 식별번호 **th**인 쓰레드의 종료시 리턴값이다.

pthread_join이 하는 일은 명확하다. 다만 주의 할것은 pthread_join은 반드시 joinable 한 상태로 생성된 쓰레드만을 기다릴 수 있다는 점이다. pthread_create로 쓰레드를 생성시킬 때, 나중에 join되지 않을 것으로 생각하고 생성시킬 수 있는데, 이렇게 되면 이 쓰레드는 종료하자마자 모든 자원을 해제하며, pthread_join으로 기다릴 수가 없다. 부모쓰레드와 떨어져서 완전히 독립적으로 작용한다고 하여, 이를 detach 한다고 한다. 쓰레드를 detach하는 방법은 아래에서 다룰 것이다.

1.8.3. pthread_mutex_init

mutex를 사용하기 위해서는 먼저 pthread_mutex_init() 함수를 이용해서, mutex 잠금 객체를 만들어줘야 한다.

이 함수는 두개의 인자를 필요로 한다.

mutex : mutex 잠금객체
mutex_attr : mutex는 **fast**, recursive, **error checking**의 3종류가 있다. 이 값을 이용해서 mutex 타입을 결정할 수 있다. NULL 일경우 기본값이 **fast**가 설정된다.
fast : 하나의 쓰레드가 하나의 잠금만을 얻을 수 있는 일반적인 형태
recursive : 잠금을 얻은 쓰레드가 다시 잠금을 얻을 수 있다. 이 경우 잠금에 대한 카운드가 증가하게 된다.
mutex_attr을 위해서 다음의 상수값이 예약되어 있다.
fast : PTHREAD_MUTEX_INITIALIZER
recursive : PTHREAD_RECURSIVE_MUTEX_INITIALIZER
error checking :
PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP

1.8.4. pthread_mutex_lock

mutex 잠금을 얻기 위한 함수다.

mutex 잠금을 얻는다라는 표현보다는 **mutex** 잠금을 요청한다라는 표현이 더 정확할 것 같다. 만약 mutex 잠금을 선점한 쓰레드가 있다면, 선점한

쓰레드가 mutex 잠금을 되돌려주기 전까지 이 코드에서 대기하게 된다.

때때로 잠금을 얻을 수 있는지만 체크하고 대기(블럭)되지 않은 상태로 다음 코드로 넘어가야할 필요가 있을 수 있을 것이다. 이 경우에는 아래의 함수를 사용하면 된다.