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

프로세스

네트워크

Sigaction

-sigaction() 함수는 signal()보다 향상된 기능을 제공하는 시그널 처리를 결정하는 함수아다 signal()에서는 처리할 행동 정보로 시그널이 발생하면 호출이될 함수 포인터를 넘겨 주었다. 그러나 sigaction()에서는 struct sigaction 구조체 값을 사용하기 때문에 좀더 다양한 지정이 가능하다.

Sigaction 시스템 호출은 특정 시그널(:12)의 수신에 대해서 취할 액션을 설정하고나 변경하기 위해서 사용한다.

signum는 시그널을 명시한다. SIGKILL(:12)과 SIGSTOP(:12)를 제외한 모든 시그널이 타당한 시그널이 될 수 있다. 만약 act 가 null 이 아니라면 signum번호를 가지는 시그널 대해 act함수가 설치된다. 만약 oldact가 null이 아니라면 sigaction구조체는 다음과 같이 정의된다.

```
struct sigaction
{
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_restorer)(void);
}
```

sa_handler : signum번호를 가지는 시그널이 발생했을 때, 실행된 함수를 설치한다. 함수외에도 SIG_DFL과 SIG_IGN을 지정 할 수 있다. 전자는 시그널에 대한 기본행동을 후자는 시그널을 무시하기 위해 사용한다.

sa_mask : sa_handler에 등록된 시그널 핸들러 함수가 실행되는 동안 블럭되어야하는 시그널의 마스크를 제공.

sg_flag : 시그널 처리 프로세스의 행위를 수정하는 일련의 플래그들을 명시한다.

```
#include <stdio.h>
#include <signal.h>

struct sigaction act_new;
struct sigaction act_old;

void sigint_handler(int signo)
{
    printf("Ctrl + C\n");
    printf("if you push it one more time then exit\n");
    sigaction(SIGINT,&act_old,NULL);
}

int main(void)
{
    act_new.sa_handler = sigint_handler;
//act_new의 시그널 핸들러 함수 설정
    sigemptyset(&act_new.sa_mask);
// mask , 특정 시그널을 막음 : 여기서는 아무것도 막지않음.
// ** 우선적으로 처리해야 할 작업에는 시그널을 막아놓는다.
```

signal (시그널번호, 시그널핸들러) 였다. Signal 함수와 같은 동작 : 성공시 0 , 실패시 -1 반환함

signum을 받으면 act 구조체 내용 실행하고, oldact 구조체에 이전 sigaction() 에서 실행된 내용을 넣고 만약 이전 실행 함수가 없다면 NULL을 넣는다.

Act, oldact는 signal.h 헤더에 정의된 struct sigaction*형 인자를 받는다.

```
    sigaction(SIGINT, &act_new, &act_old);
// act-old: 이전에 동작시켰던 시그널 핸들러 저장(없으면 null)
    while(1)
    {
        printf("sigaction test\n");
        sleep(1);
    }
    return 0;
}
```

다른 사람이 작성한 코드 쉽게 이해하는 방법
&가 들어있는것 , 즉포인터를 넣어준것은 함수내에서 값을 받아오거나 값이 변경될 수 있다.

Kill() - 시그널 전송

- kill() 함수는 쉘에서 프로세스를 죽이는 kill명령과는 달리 프로세스에 시그널을 전송한다. 프로세스에 SIGKILL을 보내면 쉘명령의 kill과 같은 역할을 한다.

- kill() 함수는 특정 프로세스 뿐만 아니라 같은 그룹 ID 가 같은 모든 프로세스에게 동시에 시그널 전송이 가능+ 권한 안에 있는 모든 프로세스에게도 시그널 전송 가능

int kill(pid_t pid, int sig);

첫번째 인자는 프로세스 id, 이프로세스 id 값에 따라 아래와 같이 처리가 분류된다. 두번쨰 인수는 시그널 번호이다.

pid	의미
양의 정수	지정한 프로세스 ID에만 시그널을 전송
0	함수를 호출하는 프로세스와 같은 그룹에 있는 모든 프로세스에 시그널을 전송
-1	함수를 호출하는 프로세스가 전송할 수 있는 권한을 가진 모든 프로세스에 시그널을 전송
-1 이와의 음수	첫번째 인수 pid 의 절대값 프로세스 그룹에 속하는 모든 프로세스에 시그널을 전송

`

헤더	signal.h
형태	**int** kill(pid_t pid, **int** sig);
인수	pid_t pid시그널을 받을 프로세스 ID **int** sig 시그널 번호
반환	0 성공 -1 실패

```c
#include<stdio.h>
#include<signal.h>

void gogogo(int voidv)
{
   printf("SIGINT Accur!\n");
   exit(0);
}

int main(void)
{
   signal(SIGINT,gogogo); // for 문 동작하다가 signal 이 들어오면 gogogo
로 간다.

   for(;;)
   {
      printf("kill Test\n");
      sleep(2);
   }
   return 0;

}

//gcc -o test test.c
// ./test &   // &으로 실행하면 greb을 사용할 필요가없음.
```

```c
#include<stdio.h>

#include<unistd.h>

#include<signal.h>

int main(int argc, char *argv[])

{

  if(argc <2)

     printf("Usage : ./exe pid\n");

  else

     kill(atoi(argv[1]),SIGINT);  // SIGINT에 숫자를 넣어라

   return 0;

}

// gcc -o kill kill.c

// ./kill pid[숫자]
```

Thread

```
#include <stdio.h>
#include <pthread.h>

void *task1(void *X)
{
    printf("Thread A Complete\n");
}

void *task2(void *X)
{
    printf("Thread B Compleate\n");
}

int main(void)
{
    pthread_t ThreadA, ThreadB;
    // Tread  생성(생성만 하고 실행x)  메모리에 올린거 아님
    pthread_create(&ThreadA, NULL, task1, NULL);
   pthread_create(&ThreadB, NULL, task2, NULL);
   // Thread 실행- 실제메모리가 올라가는 시점
   // NULL 신경 쓰지말것
    pthread_join(ThreadA,NULL);
    pthread_join(ThreadB,NULL);
}
```

스레드는 종속적이라고한다. 따라서 메모리를 고융한다

이 때문에 가장 큰 문제가 임계영역이다. 따라서 락을 걸어주고 사용한다.

//스레드를 생성한다. 1번째 입력한 스레드의 값을 뭔가 채워준다는 것이고 , 3번째인자는 구동시킬 함수 리턴은 보이드라서 어떤것이든 바꿀수있다.
// task2는 쓰레드 b 가 구동시킨다는것

//join을 하는 순간 메모리가 만들어진다.

* 컴파일시
gcc thread.c -lpthread

* void포인터:자료구조에서 재귀호출 해제하면서 push 할 때 void포인터로 받고 pop할 때 강제 캐스팅 했었음
* 병렬처리 : 스레드를 이용하여 연산량을 극대화시키는 기법

Process 종료

시작

Network 프로그래밍

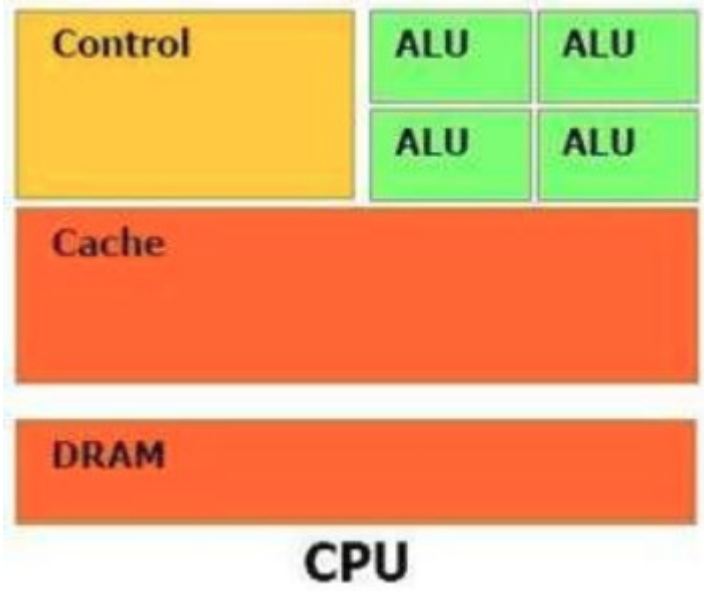
cpu는 여러개, 그래픽 카드는 만개

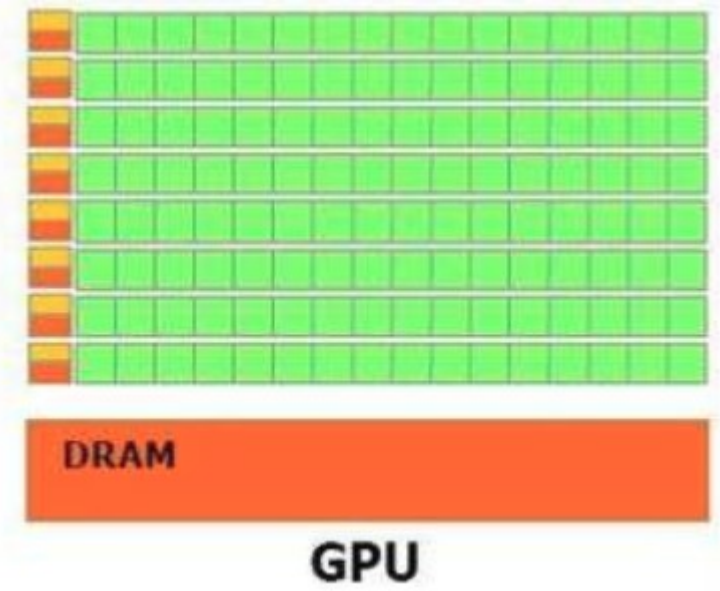
그래픽카드를 cpu 대신 사용하지 않는 이유는 클럭스피드가 떨어지기 때문이다. 하지만 cpu가 여러개 라서 bandwith 는 엉청나게 넓다.

(데이터를 한번에 입력받을 수 있는 폭, 병렬처리할수 있는능력임)

그래서 데이터를 병렬로 처리할 때는 그래픽카드가 우세하고, 고속 순차처리에는 DSP(CPU)가 우세하다.

→ 그래픽 카드는 클럭스피드가 중요하지 않다. bandwith와 개수가 중요하다.





→ alu는 연산을 담당함

[1] CS (cilent, server)

[2] 토폴로지

-위상수학은 제외

- 네트워크 요소들 (링크,노드) 을 물리적으로 연결해 놓은것, 구성도를 말함.

- 그래픽알고리즘

[3] TCP/IP 프로토콜

 이론– OSI 7 layer 버클리에서 7계층이 너무 많다고 4계층을 만들었는데 이를 tcp/ip 프로토콜이라한다. 유닉스, 리눅스에 최적화 되어 있음

Ifconfig 를 쳐서 봐라

256 게이트웨이

255 브로드캐스트

254

ip의 종류는 2가지가 있다

1. 공인 ip – wan 통신을 하기 위해서는 필요하다 (네이버나 다음에 접속할 때 필요함.)
2. 사설 ip – 공유기

→ 공인 ip는 하나인데 여러명이 인터넷을 사용할 수 있는 이유는 NAT프로토콜 덕분이다.

mac 통신

랜카드의 고유 번호, 인터넷으로 하드웨어에 접속하게되면 mac주소가 남게된다.
mac주소는 스위치 장비에 남는다. 스위치 장비에 남게되는 이유는 mac주소를 보고 어디로 보내줘야 할지 정하기 때문이다. ip주소를 보고 보내주는 것은 바로 라우터이다.
스위치에 mac을 찾고 싶다고 요청하면 broadcasting한다. (ddos공격이 가능함)

스위치가 본인의 맥테이블을 검색하여 요청받은 맥을 찾지 못하면 스위치가 관리하는 대역이 아니므로 라우터에 ip를 보내 경로를 물어본다. 해당 ip를 가진 장치가 이를 인식하면 통신이 이뤄진다
→ 이과정은 IPC이다 . 즉, 네트워크는 = 원격 ipc이다

네트워크간 통신은 가능하다? Ipc 의 경우 그렇다.
우리가 네이버라는 인터넷창을 보고 있는 것도 웹서버가 정보를 보내줘서 가능한것이다.

```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;

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

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

    int sock;
    int str_len;
    si serv_addr;
    char msg[32];

    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");

    str_len = read(sock, msg, sizeof(msg) -1);

    if(str_len == -1)
        err_handler("read() error!");

    printf("msg from serv: %s\n", msg);
    close(sock);

    return 0;
}
```

```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;

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

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

    si serv_addr;
    si clnt_addr;
    socklen_t clnt_addr_size;

    char msg[] = "Hello Network Programming";

    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);
    clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr, &clnt_addr_size);

    if(clnt_sock == -1)
        err_handler("accept() error");

    write(clnt_sock, msg, sizeof(msg));
    close(clnt_sock);
    close(serv_sock);

    return 0;
```

```
#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;

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


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

    int sock;
    int str_len;
    si serv_addr;
    char msg[32];


    if(argc !=3){ //argc 가 3개 ip주소를 담는다.  사설 ip주소는 무조건 192.168.0.x임
        printf("use: %s <IP> <port>\n", argv[0]);
        exit(1);
    }

    sock = socket(PF_INET, SOCK_STREAM, 0);

    if(sock == -1)
```

```
        err_handler("socket() error");

// serv_addr  초기화 패턴
// &serv_addr은 si구조체 임
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;                 // TCP 사용
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]); //아이피 주소
    serv_addr.sin_port = htons(atoi(argv[2]));      // 포트번호

// 서버와 connect → 서버의 listen  때 accept하여 서버와 클라이언트간 통신이가능해짐
//sock: 자기 자신에 대한 네트워크 파일 디스크립터
//&serv_addr 에 연결
    if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("connect() error");

// 서버에 쓰기를 했으므로 이제 read함
// read는 블록킹 함수 들어올 때까지 움직이지 않음
//파일 디스크립터(sock)에서 읽고,  msg에 저장함.

     str_len = read(sock, msg, sizeof(msg) -1);


    if(str_len == -1)
        err_handler("read() error!");

    printf("msg from serv: %s\n", msg);
    close(sock);

    return 0;
}
```

Server

```
#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;

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


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

    si serv_addr;
    si clnt_addr;
    socklen_t clnt_addr_size;

    char msg[] = "Hello Network Programming";

    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);   //자기자신을 주소로 받음 로컬호스트
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)  //5명 받겠다. 그이상은 x
    err_handler("listen() error");

clnt_addr_size = sizeof(clnt_addr);
clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr, &clnt_addr_size);

if(clnt_sock == -1)
    err_handler("accept() error");

write(clnt_sock, msg, sizeof(msg));
close(clnt_sock);
close(serv_sock);

return 0;
    err_handler("connect() error");

str_len = read(sock, msg, sizeof(msg) -1);

if(str_len == -1)
```

```
        err_handler("read() error!");

    printf("msg from serv: %s\n", msg);
    close(sock);

    return 0;
}
```

Socket은 파일이다 socket.c

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/socket.h>

int main(void)
{
   int fd[3];
   int i;

   fd[0] = socket(PF_INET, SOCK_STREAM, 0);
   fd[1] = socket(PF_INET, SOCK_DGRAM, 0);
   fd[2] = open("test.txt",O_CREAT | O_WRONLY | O_TRUNC);

   for(i = 0; i < 3; i++)

      printf("fd[%d] = %d\n", i, fd[i]);
   for(i = 0; i < 3; i++)
      close(fd[i]);
   return 0;

}
```

: 3,4,5 가 잘나온다.
이는 소켓도 파일디스크립터를 만든다라고 볼 수 있다.

// STREAM – tcp
// DGRAM - udp

오늘도 띵언이 나옵니다.

모두 파일이다.

read_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;

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

int main(int argc, char **argv){
int sock;
int str_len = 0;
si serv_addr;
char msg[32] = {0};
int idx = 0, read_len =0;
```

```
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)) == 01)
err_handler("connect() error");

while(read_len = read(sock, &msg[idx++],1)){
if(read_len == -1)
err_handler("read() error!");

str_len += read_len;
}// read함수로 read_len을 체크하고 있다.

printf("msg from serv : %s\n",msg);
printf("read count : %d\n",str_len);
close(sock);
```

// connect와 read함수 때, 넘어올 데이터의 길이 등 모든 정보가 들어온다.

//data가 중간에 끊겼을 때, 손실을 방지한다.

return 0; }	

convert_endian.c

```
include <stdio.h>


//  변수는 메모리에 저장하는 공간
int main(void)
{
    unsigned short host_port = 0x5678;
    unsigned short net_port;
    unsigned long host_addr = 0x87654321;
    unsigned long net_addr;


    net_port = htons(host_port);
    net_addr = htonl(host_addr);


    printf("Host Ordered Port: %#x\n",host_port);
    printf("Network ordered port: %#x\n",net_port);
    printf("Host Ordered Address: %#lx\n", host_addr);
    printf("Network Ordered Address: %#lx\n",net_addr);
// 크로스 매칭을 하는 이유 컴퓨터마다 받아드리는 순거가 다름 그래서 꼬이지말라고 바꿔주는것
```

```
    return 0;
}
```

```
lee@lee-Lenovo-YOGA-720-13IKB:~/my_proj/Homework/sanghoonlee/lec/lhs/linux_system/3_29$ ./a.out
Host Ordered Port: 0x5678
Network ordered port: 0x7856
Host Ordered Address: 0x87654321
Network Ordered Address: 0x21436587
```

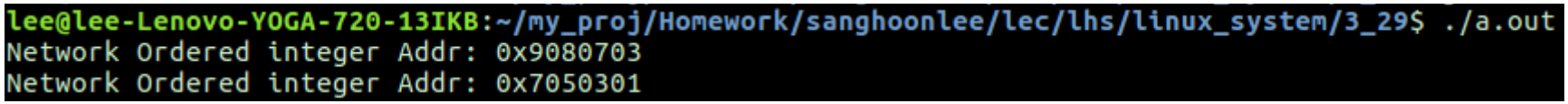
inet_addr.c

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

int main(int argc, char **argv)
{
   char *addr1 = "3.7.8.9";  // 빅엔디안, 리틀엔디안 서로다른형식으로 받으면 어
떻게 될까연
   //메모리에 09 08 07 03 으로 저장 --> 빅엔디안 : 순서대로 받는다. 03 07 08 09
   //                          --> 리틀엔디안 : 09.05.07.03 으로 받는다
   char *addr2 = "1.3.5.7";

   unsigned long conv_addr = inet_addr(addr1);
   if(conv_addr == INADDR_NONE)
      printf("Error!\n");
   else
      printf("Network Ordered integer Addr: %#lx\n",conv_addr);

   conv_addr = inet_addr(addr2);
   if(conv_addr == INADDR_NONE)
      printf("Error!\n");
   else
      printf("Network Ordered integer Addr: %#lx\n",conv_addr);
   return 0;
}
// 애초에 메모리에 들어가는 형식으로 변환 후 전달하는 개념
```



보드 http://www.ti.com/tool/TMDSEVM572X#1

/argv

:set Hlserch

argv 를 전부찾아줘.