TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/4/5
수업일수	31 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

1. 파일나누기 - 네트워크

- init_sock.h

```
#ifndef __INIT_SOCK_H__
#define __INIT_SOCK_H__

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

typedef struct sockaddr_in              si;
typedef struct sockaddr * sp;

void err_handler(char *msg); //에러메세지 출력 함수
int init_sock(void); //소켓 생성 함수
void init_sock_addr(si *, int, char **, int); //주소 지정함수
void post_sock(int, si *, int); //bind, listen 설정 함수

#endif
```

- common.h

```
#ifndef __COMMON_H__
#define __COMMON_H__

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>

typedef struct sockaddr_in		si;
typedef struct sockaddr * sp;

typedef struct __d{
		int data;
		float fdata;
} d;

#define BUF_SIZE				32

#endif
```

-init_sock.c

```
#include "init_sock.h"

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

int init_sock(void)
{
		int sock;

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

```
	if(sock == -1)//소켓에러
		err_handler("socket() error!");

	return sock;//소켓반환
}

// serv = 0, clnt = 1
void init_sock_addr(si *serv_addr, int size, char **argv, int opt)
{
	memset(serv_addr, 0, size);//serv_addr 초기화
	serv_addr->sin_family = AF_INET;//family 설정

	if(opt)//opt 1을 주면 클라이언트
	{
		serv_addr->sin_addr.s_addr = inet_addr(argv[1]);//addr 설정
		serv_addr->sin_port = htons(atoi(argv[2]));//port 설정
	}
	else//opt 0을주면 서버
	{
		serv_addr->sin_addr.s_addr = htonl(INADDR_ANY);//addr 설정 서버 본인이니까
INADDR_ANY
		serv_addr->sin_port = htons(atoi(argv[1]));//port 설정
	}
}

void post_sock(int serv_sock, si *serv_addr, int size)
{
	if(bind(serv_sock, (sp)serv_addr, size) == -1)//bind 설정
		err_handler("bind() error!");
```

```
	if(listen(serv_sock, 5) == -1)//listen 설정
		err_handler("listen() error!");
}
```

-basic_serv.c

```
#include "common.h"
#include "init_sock.h"

int main(int argc, char **argv)
{
	int serv_sock, clnt_sock;
	si serv_addr, 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 = init_sock(); //소켓설정
	init_sock_addr(&serv_addr, sizeof(serv_addr), argv, 0);//addr 설정
	post_sock(serv_sock, &serv_addr, sizeof(serv_addr));//bind 랑  listen  설정

	clnt_addr_size = sizeof(clnt_addr);//clnt 주소사이즈
	clnt_sock = accept(serv_sock, (sp)&clnt_addr, &clnt_addr_size);//accept

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

```
	write(clnt_sock, msg, sizeof(msg));//받은 메세지 출력

	close(clnt_sock);
	close(serv_sock);

	return 0;
}
```

-basic_clnt.c

```
#include "common.h"
#include "init_sock.h"

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

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

	sock = init_sock();//소켓생성
	init_sock_addr(&serv_addr, sizeof(serv_addr), argv, 1);//addr 설정

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

	len = read(sock, msg, sizeof(msg) - 1);//데이터 읽기
```

```
	if(len == -1)//read 에러
		err_handler("read() error!");

	printf("msg from serv: %s₩n", msg);//데이터 쓰기
	close(sock);

	return 0;
}
```

2.웹 서버 띄우기

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

#define BUF_SIZE		1024
#define SMALL_BUF	100

typedef struct sockaddr_in		si;
typedef struct sockaddr * sp;

void error_handling(char *msg) //에러메세지 출력
{
	fputs(msg, stderr);
```

```
		fputc('₩n', stderr);
		exit(1);
}

void send_error(FILE *fp) //모니터에 에러 출력 함수
{
		char protocol[] = "HTTP/1.0 400 Bad Request₩r₩n";
		char server[] = "Server : Linux Web Server₩r₩n";
		char cnt_len[] = "Content-length:2048₩r₩n";
		char cnt_type[] = "Content-type:text/html₩r₩n₩r₩n";
		char content[] = "<html><head><title>Network</title></head>" "<body><font size=+5><br>
오류발생! 요청 파일명 및 방식 확인!" "</font></body></html>";
		fputs(protocol, fp);
		fputs(server, fp);
		fputs(cnt_len, fp);
		fputs(cnt_type, fp);
		fflush(fp);
}

char *content_type(char *file)
{
		char extension[SMALL_BUF];
		char file_name[SMALL_BUF];
		strcpy(file_name, file);
		strtok(file_name, ".");
		strcpy(extension, strtok(NULL, "."));

		if(!strcmp(extension, "html") || !strcmp(extension, "htm"))
			return "text/html";
		else
			return "text/piain";
```

```
}

void send_data(FILE *fp, char *ct, char *file_name)
{
	char protocol[] = "HTTP/1.0 200 OK₩r₩n";
	char server[] = "Server:Linux Web Server₩r₩n";
	char cnt_len[] = "Content-length:2048₩r₩n";
	char cnt_type[SMALL_BUF];
	char buf[BUF_SIZE];
	FILE *send_file;

	sprintf(cnt_type,"Content-type:%s₩r₩n₩r₩n", ct);
	send_file = fopen(file_name, "r");
	if(send_file == NULL)
	{
		send_error(fp);
		return;
	}

	fputs(protocol, fp);
	fputs(server, fp);
	fputs(cnt_len, fp);
	fputs(cnt_type, fp);

	while(fgets(buf, BUF_SIZE, send_file) != NULL)
	{
		fputs(buf, fp);
		fflush(fp);
	}
	fflush(fp);
	fclose(fp);
```

```
}

void *request_handler(void *arg)
{
	int clnt_sock = *((int *)arg);
	char req_line[SMALL_BUF];
	FILE *clnt_read;
	FILE *clnt_write;

	char method[10];
	char ct[15];
	char file_name[30];

	clnt_read = fdopen(clnt_sock, "r");
	clnt_write = fdopen(dup(clnt_sock), "w");
	fgets(req_line, SMALL_BUF, clnt_read);

	if(strstr(req_line, "HTTP/") == NULL)
	{
		send_error(clnt_write);
		fclose(clnt_read);
		fclose(clnt_write);
		return ;
	}

	strcpy(method, strtok(req_line, " /"));
	strcpy(file_name, strtok(NULL, " /"));
	strcpy(ct, content_type(file_name));

	if(strcmp(method, "GET") != 0)
	{
```

```
                send_error(clnt_write);
                fclose(clnt_read);
                fclose(clnt_write);
                return ;
        }

        fclose(clnt_read);
        send_data(clnt_write, ct, file_name);

}

int main(int argc, char **argv)
{
        int serv_sock, clnt_sock;
        si serv_addr, clnt_addr;
        int clnt_addr_size;
        char buf[BUF_SIZE];
        pthread_t t_id;

        if(argc != 2) //인자 2개가 아닐 때 에러 출력
        {
                printf("Use: %s <port>₩n", argv[0]);
                exit(1);
        }

        serv_sock = socket(PF_INET, SOCK_STREAM, 0); //소켓 생성
        memset(&serv_addr, 0, sizeof(serv_addr)); // 서버 주소 초기화 및 설정
        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) //bind 설정 및 에러
		error_handling("bind() error");
	if(listen(serv_sock, 20) == -1) //접속자 20명 받음
		error_handling("listen() error");

	for(;;)
	{
		clnt_addr_size = sizeof(clnt_addr);
		clnt_sock = accept(serv_sock, (sp)&clnt_addr, &clnt_addr_size);
		printf("Connection Request: %s:%d₩n", inet_ntoa(clnt_addr.sin_addr),
ntohs(clnt_addr.sin_port));//클라이언트 주소 및 포트번호 출력
		pthread_create(&t_id, NULL, request_handler, &clnt_sock);//request_handler 수행하는
함수 쓰레드 생성
		pthread_detach(t_id);
	}

	close(serv_sock);

	return 0;
}
```

-first.html

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>
</body>
```

</html>

~결과
xeno@xeno-NH:~/proj/0405$ gcc -o web_serv web_serv.c -lpthread
xeno@xeno-NH:~/proj/0405$ ./web_serv 7770
Connection Request: 127.0.0.1:41334

접속한 아이피가 뜬다.

Localhost:7770/first.html 을 인터넷 주소창에 치면 위의 fist.html 에서 썼던 내용들이 출력된다.

- 홀펀칭(Hole Punching)
NAT 는 사설아이피를 만든다. 사설아이피를 만든 NAT 들이 통신할 때 홀펀칭이 필요함.

3. Linux Kernel

-Chaptor 0 운영체제 이야기

운영체제(OS) : 유저가 컴퓨터를 사용할 수 있도록 서포트 해주고, CPU, 메모리, 디스크 등의 자원(HW+SW)을 관리하는 자원관리자.

OS 는 디스크. CPU, 메모리를 관리하는데 어떠한 프로그램을 작성하여 저장하려 할 때 저장되는 내용은 파일이라는 객체로 관리되어 비휘발성 저장매체인 디스크에 저장된다.

Disk : 보조 저장장치, 디스크의 추상화는 파일이다.

CPU : 컴퓨터 처리기, cpu 의 추상화는 프로세스이다.

메모리 : 하드웨어의 추상화된 소프트웨어적 개념이 페이지이다. 페이지는 구조체로 되어있음.

<-> 메모리의 최소단위는 페이지 프레임으로 4KB(4096byte)이다.

만약 유저가 작성한 프로그램을 저장하려 할 때 file 이라는객체(task_struct.files_struct.FILE 구조체)로 관리되고 작성한 프로그램을 컴파일 하게 되면 실행파일이 생성된다. 그 실행파일은 디스크에 저장되게 되는데 inode 도 함께 공간을 할당해주게 된다. inode 라는 것은 디스크에서 파일을 찾기 위한 정보인 디스크 블록의 위치, 만들어진 시간, 만든 사람, 접근제어 정보(rwx) 등을 알려준다.

inode 는 디스크 블록을 여러 개 가리킬 수 있는데 메모리의 최소단위인 4KB 는 물리메모리의 최소단위를 디스크도 따르기 때문에 5KB 같은 크기의 데이터를 저장할 때 두 개의 블록에 저장하여 inode 가 여러 개를 가리킬 수 있다. Stat()이라는 함수를 사용할 때 inode 의 데이터를 이용해서 알려줌.

컴파일 후 inode 와 디스크 블록을 잇고 디스크에 저장한 뒤 실행파일을 수행하면 task 라는 새로운 객체가 생성된다. 이 task 라는 객체들은 멀티태스킹을 하기 위해 context switching 을 하며 CPU 를 점유하려 하는데, CPU 자원을 공평하게 나누어 주는 방식을 라운드-로빈(round-robin)이라는 방식이 있다. 우선순위가 없이 같은 시간이 할당되는 방식이다.

CPU 는 디스크에 직접 접근하지 못하여 디스크의 내용이 메모리로 적재되어야한다. task 라는 객체는 task_struct 가 만들어지고 Demand on paging 과 swapping 이라는 방식을 이용하여 큰 데이터를 메모리에 올린다. 쓰레드나 프로세스 생성하면 tast_struct 가 메모리에 올라가게 된다. 각 태스크는 세그먼트 테이블과 페이지 테이블을 이용하여 자신에게 할당된 페이지 프레임을 관리한다. 세그먼트 테이블은 컴파일러 레벨에서 stack, data, heap, text 같은 것이고 가상 메모리 레이아웃에 해당한다. 이 세그먼트와 페이지 프레임을 통해 가상메모리에서 물리메모리로 올리는 페이징이 가능하다.

-(숙제) 운영체제 동작 비유

(비유 1) - fork( ), 데몬 프로세스

fork( )

: fork 한 밑의 소스코드부터 복제한다.

: 성공 시 자식의 pid 값을 반환한다.

: 프로세스가 실행될 때 부모 프로세스가 먼저 실행될 수 있고 자식 프로세스가 먼저 실행될 수 도 있다.

: fork 하여 부모 프로세스와 자식 프로세스가 만들어 졌을 때 부모 프로세스에서의 fork( )의 반환 값은 자식 프로세스이고, 자식 프로세스에서의 fork( )의 반환 값은 0 이게 된다. 새로 만들어진 프로세스는 fork 로 새로운 프로세스를 만들지 못하기 때문이다.

: fork 를 통하여 multitasking 을 할 수 있다. 이전에 했던 pipe 통신은 read 함수가 blocking 이어서 한쪽에서만 메시지 전송이 가능했는데 non-blocking 으로 설정하지 않아도 fork 를 이용하여 또 다른 프로세스를 만들게 되면 context switching 되며 multitasking 이 구현 가능하다. 만약 무한 반복하는 프로그램이 fork 되어 자식과 부모 프로세스가 있을 때, 각 프로세스마다 할당된 시간에 따라 한 프로세스 씩 처리되고 할당 시간 내에 처리하지 못한 프로세스는 wait queue 에 대기하고 있다가 run queue 에서 실행되던 프로세스가 내려오면 우선순위에 따라 run queue 에 올라가게 된다.

데몬 프로세스

: 일반 프로세스 같은 경우는 터미널을 끄게 되었을 때 프로세스가 죽게 되는데 데몬 프로세스는 터미널을 꺼도 죽지 않는 프로세스를 말한다.

: 터미널에서 ' ps -ef | grep a.out ' 명령어를 치면 실행되는 프로세스들이 뜨게 되는데 데몬 프로세스는 시간 옆에 pts/7, pts/4 등의 정보가 뜨는 구간에 ' ? '가 뜨게 된다.

: 데몬 프로세스를 응용하여 SIGINT 나 SIGQUIT, SIGKILL 등의 시그널을 무시하는 프로그램을 작성했을 때 실행하게 되면 Ctrl+c 나 Ctrl+q 는 실행되지 않지만 kill -9 명령어는 무시하지 못한다.

(비유 2) – multitasking, context switching

multitasking

: CPU 는 한번에 하나의 작업밖에 하지 못한다. 하지만 ps -ef 명령어로 실행되는 프로세스들을 보면 여러 프로세스가 동시 다발적으로 실행되는 것을 확인할 수 있는데 이는 context switching 을 통해 multitasking 이 가능한 것이다.

: 프로세스들은 CPU 를 점유하기 위해 메모리에 올라오게 된다. 메모리에 모든 데이터가 올라오는 것은 한계가 있어 Demand on paging 과 swapping 기법을 사용한다.

: 모든 CPU 에는 Wait Queue 와 Run Queue 가 존재한다. 프로세스가 메모리에 올라와 실행될 때 Run Queue 에 올라오게 되고 할당된 시간 내에 프로세스를 마치지 못하면 Wait Queue 에 올라오게 된다.

context switching

: 프로세스들이 Run Queue 와 Wait Queue 를 왔다 갔다 하면, 실행되던 중간에 해당 프로세스를 동작하지 못하였다가 다시 Run Queue 에 올라 제어권을 얻게 되었을 때 이전에 실행되려는 부분을 이어서 수행하도록 하는 것이 context switching 이다.

: 우선순위에 따라 context switching 의 순서와 할당시간이 달라진다.

(비유 3) - IPC, signal, pid, block vs non-block

IPC

: 프로세스는 독립적이어서 메모리를 공유하지 못해 pipe 나 message queue, shared memory 를 이용하여 메모리를 공유한다.

signal

: 비동기 처리일 때 행동지침을 정해 놓는다. 어떤 시그널이 발생하면 기존 방법으로 처리할지, 무시할지, 프로그램에서 직접 처리할지를 설정할 수 있다.

blocking VS non-blocking

: fork 를 이용하지 않고 pipe 통신을 하게 되면 read 함수는 blocking 이어서 작성한 소스코드 순서에 따라 보내야 하는데 non-blocking 은 순서에 상관없이 pipe 통신을 할 수 있다.

: 다수가 빠르게 통신할 때는 non-blocking 이 좋고 순차적으로 진행되어야 할 때는 blocking 이 좋다.

(비유 5) - swap

swap

: 가상 메모리에 값을 저장하려면 메모리의 user 영영의 가짜 주소를 할당 받아 10bit/10bit/12bit 로 나누어 12bit 짜리는 물리메모리에 바로 저장하고 프로세스가 실행될 때 생성되는 tast_struct 내에 있는 배열?에 첫번째 10bit 를 저장하고 그 배열에는 두번째 10bit 가 저장된 주소 값을 포함하고있다. 두번째 배열에도 마찬가지로 두번째 10bit 가 저장 되어있고 12bit 가 저장된 블록의 위치가 저장 되어있다. 이러한 방식으로 메모리가 채워지는데 메모리가 부족해질 경우 디스크를 이용한 swap 방식을 사용한다.

: swap 은 사용했던 메모리를 활용하고 자주 쓰고, 없으면 안되는 것들을 queue 형식으로 저장한다. 필요할 때 마다 올리고 내린다. 메모리 계층구조에서 메모리의 속도가 느리니 현재 필요한 정도만을 페이징 하는 것을 demand on paging 이라고 한다.