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

#31

2018.04.05

강사:Innova Lee(이 상훈)

학생: 김시윤

과제

비유 1.

얼마 전 회사를 차린 A 씨는 요즘 행복한 고민에 빠졌다. 처음 회사를 차릴 때 우려했던 바와는 달리 회사는 날로 번창하여 수입은 계속 늘어났지만, 이에 비례해서 일거리도 계속 늘어나면서 더 이상 모든 일을 혼자 처리하기엔 역부족이었기 때문이었다.
어쩔 수 없이 A 씨는 회사의 운영을 책임질 사장을 뽑기로 결정한다. 그래서 A 씨는 인력회사에 전화하여 컴퓨터 활용과 외국어 구사가 가능하며 성실한 사람을 보내달라고 요청하였다.
그런 뒤 A 씨는 새로 회사에 출근할 사람을 위해 책상과 의자, 그리괘 사무를 보는데 필요한 기타 용품을 준비해 두었다. 다음날, A 씨의 회사로 출근한 B 씨는 A 씨로부터 어떻게 회사를 끌어가라는 지시를 받았다.
그러자 B 씨는 원래 A 씨가 했었던 수많은 귀찮은 일들을 모두 처리해 주었다. A 씨는 너무도 편했고, 이에 따라 자신 본연의 임무에 충실할 수 있었다.
B 씨는 A 씨가 시킨 일들을 모두 완수하기 위해 C 씨와, D 씨, C 씨와 D 씨는 평소 할 일이 없을 땐 그저 사무실에서 쉬고 있었다. 그러나 C 씨는 누가 일을 시키지 않더라도 회사로 전화가 걸려오면 언제든 그 전화를 받아 응대하는 일을 수행하였고, D 씨는 A 씨가 시키는 일이 있을 때 마다 그 일을 수행하였다.
A 씨의 회사가 날로 번창하자 B 씨는 더 많은 부하직원을 채용하기로 결심하였다. 우선 C 씨의 업무가 너무 과중한 것 같아 C 씨와 동일한 일을 수행하는 C2 씨와, 제품을 조립하는 E 씨, 조립된 제품을 포장하는 F 씨, 포장된 제품을 발송하는 G 씨를 추가로 채용하였다.

C, C2, D, E, F 그리고 G 씨는 한정된 사무실 공간에 있었지만 E 씨가 제품을 조립해야 할 때에는 남들보다 더 많은 공간을 쓸 수 있도록 배려하였다. 또한 제품을 포장해야 할 때에는 F 씨가, 제품을 발송할 때는 G 씨가 공간을 더 쓸 수 있도록 B 씨가 항상 공간 배정을 해 주었기에 모두들 사무실이 좁다는 생각을 하지 않고 자신의 맡은바 임무를 완수할 수 있었다.

비유 1 을 Demand on paging 을 이야기 하는것 같다.

여기서 A 씨는 User 고 B 씨는 user 가 커널에서 만든 프로세스라고 볼수있다.
프로세스가 실행이 되게되면 task_struct 가 생긴다.
즉 메모리 공간이 생긴다는것이다 여기서 작업을 수행할 C C2 가 데몬인데 그밑에 D E F 가 두개씩있다. 이러면 메모리 공간을 차지하는 비율이 매우 커져 Demand on paging 을 하고 그것을 하드디스크 Swap 이 서포트한다. 따라서 메모리에서 중요한 C, C2 는 항상 작업을 하고 별로 필요없는것들인 D,E,F 까 하드디스크 큐에 저장되어 있다가 필요할때 꺼내서 쓰는 방식인거 같다.

이 얘기를 게임서버를 예로들어서 설명했으며 이렇게하면 로딩시간이 빠르고 효율적이라고 해주셨었다.

```
fork();

if(pid > 0)
{
    exit(0);
}
else if(pid == 0)
{
  E(a,b);
  F(a,b);
  G(a,b);
    fork();
     if(pid2>0)
     {
        exit(0);
      }
      else if(pid2 ==0)
   {
    E(a,b);
```

```
    F(a,b);
    G(a,b);
    }
}
```

이 이상생활을 코드로 나타내면 이런식으로 표현할수 있을거같습니다..
동일한 일을하는 데몬이 2 개에 E F G 함수가 동시에 호출되어있습니다.
pid 를 배열로 만들면 포크를 한번 쓰고도 가능할거같습니다.
그런데 여기서 궁금한게 포크를 하게되면 task_struct(사무실) 이 여러개 생기게되고 프로세스가 만들어져 데이터 공간이 늘어나는게 아닌지 궁금합니다. 만약 데이터를 저장할 수 있는 공간이 늘어나는게 아니라면 일정한 데이터 공간에서 포크를 하면 메모리 공간에 비해 요구 공간이 많을것이고 그러면 디멘드 온 페이징을 하는게 현명하다는 얘기를 하는거 같습니다.

비유 2

카사노바 박씨에겐 24 명의 여자 친구가 있다. 모두 내일 꼭 만나달라고 아우성이었다. 대체 내일 누굴 만날 것인가?

친구들에게 물어보았다. 하지만 친구들마다 다른 충고를 해주었다. 친구 이씨는 가장 예쁜 여자를 만나겠다고 하였고, 또 다른 친구 김씨는 가장 착한 여자를 만나겠다고 하였다.

하지만 진정한 카사노바 박씨는 어떻게 하면 더 많은 여자 친구를 만날 수 있을까 고민하던 중, 24 명의 여자 친구를 가장 예쁜 순서대로 줄을 서게 하고 순서대로 정해진 시가ㄴ 만큼씩만 만나기로 하였다. 따라서 단 한명의 여자 친구만 카사노바 박씨를 만나는 상태이고, 다른 모든 여자 친구는 박씨를 기다리는 상태이다.

그런데 문제가 생겼다. 얼마나 오랫동안 만날 것인가? 1 시간으로 고정하면 어떨까?
할 얘기가 많은 여자 친구와도 1 시간, 할 얘기가 별로 없는 여자 친구와도 1 시간 만난다는 것은 비효율 적이었다. 그래서 카사노바 박씨는 할 얘기가 많은 여자 친구와는 보다 더 오랜 시간 만나기로 결정하였다.

또 다른 문제가 생겼다. 오늘 새로 알게 된 여자 친구가 급히 만나달라고 조르기 시작했다. 카사노바 박씨는 고민에 빠졌다. 여자 친구를 만나던 중 다른 여자 친구가 급히 만나자고 하면 응할 것인가 말 것인가. 캇나봐 박씨는 급한 일이 있는 경우에만 다른 여자 친구를 만나기로 결정하였다.

그렇게 여러 여자 친구를 만나고 있던 카사노바 박씨는 여러 여자친구를 만나고 있는 사실을 들키지 않게 치말한 준비를 하였다. 그래서 박씨는 여러 여자친구를 만나더라도 했던 얘기를 또 하는 등의 실수를 범하지 않을 수 있었다.

비유 2 는 Multi tasking 과 SystemCall 두개가 합쳐져있다.

Runqueue 에는 1 개의 프로세스가 들어있고 Wait queue 에 23 개의 프로세스가 있다.
거기서 이쁘고 할말많은 프로세스들 순서데로 들어있다.
작업을 하는도중 시그날 또는 시스템콜이 날라오게되면 중간에 요청을 수행하고
다시 하던 작업으로 돌아간다.

```
Fork();
for(;;)
{
 if(i==a || i==A)
   write(1,"SystemCall",11);

if(pid>0)
{
  for(i=A;i=Z;i++)
 printf("%c",i);
}else if(pid ==0)
{
  for(i=a;i=z;i++)
 printf("%c",i);
}
}
```

이런식으로하면 context switching 과 시스템콜의 융합을 볼수 있다.

비유 3

철수는 오늘도 영희가 만나고 싶었다. 그래서 철수는 영희네 집 앞으로 찾아가 만낟라라고 영희 방 창문에 돌을 던졌다.
돌을 던지면 창문은? 기본적으로 깨진다.
창문이 ㄲ재니느 것이 싫다면, 영희는 누군가 돌 던질 때를 대비해서 항상 받을 준비를 하고 있어야 한다.

그런데 영희는 갑자기 궁금해졌다. 철수가 왜 돌을 던졌을까? 보고 싶으니 나오라고? 채팅하고 싶으니 메신져에 들어오라고? 아님 화가 나서 그냥? 그래서 철수와 영희는 왜 돌을 던졌는지를 나타내는 번호를 돌에 쓰기로 결정하였다.
한편 철수가 던진 돌을 받은 영희는 만나자는 번호가 써 있음을 알게 되었다. 그래서 세수도 하고 옷도 갈아입고, 만날 준비를 하고 있었다. 이때 철수가 만나달라고 또 돌을 던지면 어떨까? 준비하고 있는데 또 돌을 던지니 매우 짜증 날 것이다. 그래서 영희는 '나갈 준비하고 있으니 또 돌 던지지 마시오' 라고 창문에 써서 붙여 놓기로 하였다.
이
돌을 통해 간단한 번호라는 정보를 주고 받던 철수와 영희는 문득 돌을 던지는 것이 빠르고 간단하긴 하지만 자세한 정보를 주고 받을 수 없다는 생각을 하게 되었다. 편지를 쓰면 어떨까? 전화를 하는 건 어떨까?

위에 돌을 던지는 행위는 signal 을 의미한다 우리는 일전에

WIFEXITED(status) :정상 종료일 때 참

WEXITSTATUS(status): 정상 종료일 때 하위 8 비트 값

WIFSIGNALED(status): 시그널에 의해 종료했을 때 참

WTERMSIG(status): 시그널에 의해 종료했을 때 시그널 번호

WCOREDUMP(status): 코어 파일 발생 여부

WIFSTOPPED(status): 일시 중단 상태일 때 참

WSTOPSIG(status): 일시 중단시킨 시그널 번호

이와같이 시그날 상태를 확인하는 방법을 배웠었다.
돌에 적힌 숫자는 이 상태를 얘기한다.
sigprocmask(SIG_BLOCK, 돌, NULL)
시그날을 블록해준다. 아직 다시 복습해야할것같다 ..

시그널은 한가지 작업밖에 하지못해 데이터 전송에 효율적이지 못하다.
그래서 IPC 가 나왔고 발전해 sock 통신이 나왔다..

배운내용 복습

---소스코드 기능별 파일 분리---

file_server 와 file_client 를 이용하여 기능 분리 구현을 해보았다.

basics.h

기초가 되는 헤더파일이라 이름을 배이직이라 정했다.

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

typedef struct sockaddr_in si;
typedef struct sockaddr * sap;

#define BUF_SIZE 32

int init_sock(void);
void init_sock_addr(si *, int, char **, int);
void post_sock(int, si *, int);
void err_handler(char *msg);
void write_clnt(int fd, char *buf,int clnt_sock);
void recv_file(int fd, char *buf,int sock);
```

init.c

sock 을 할당하는 초기 서버 클라이언트 설정

```
#include "basics.h"

void err_handler(char *msg)
{
        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)
        err_handler("socket() error!");

    return sock;
}

// serv = 0, clnt = 1
void init_sock_addr(si *serv_addr, int size, char **argv, int opt)
{
    memset(serv_addr, 0, size);
    serv_addr->sin_family = AF_INET;
//일반적인 인터넷 프로토콜 쓴다.(TCP)

    if(opt)
    {
//클라이언트
        serv_addr->sin_addr.s_addr = inet_addr(argv[1]);//아이피 셋팅
        serv_addr->sin_port = htons(atoi(argv[2]));//포트번호셋팅
    }
    else
    {
//서버
        serv_addr->sin_addr.s_addr = htonl(INADDR_ANY);//자기자신
```

```
        serv_addr->sin_port = htons(atoi(argv[1]));//port num
    }
}

void post_sock(int serv_sock, si *serv_addr, int size)
//서버가 구동할때 바인드 리슨
{
    if(bind(serv_sock, (sap)serv_addr, size) == -1)
        err_handler("bind() error!");

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

write_clnt.c

서버에서 클라이언트에 파일을 써주는 기능의 함수.

```
#include "basics.h"

void write_clnt(int fd,char *buf,int clnt_sock)
{
    int read_cnt;
    for(;;)
    {
        read_cnt = read(fd, buf, BUF_SIZE);
        if(read_cnt < BUF_SIZE)
        {
            write(clnt_sock, buf, read_cnt);
            break;
        }
        write(clnt_sock, buf, BUF_SIZE);
    }

    shutdown(clnt_sock, SHUT_WR);
    read(clnt_sock, buf, BUF_SIZE);
    printf("msg from client: %s\n", buf);

    close(fd);
}
```

recv_file.c

//클라이언트에서 서버의 데이터를 받기위함 함수

```
#include "basics.h"

void recv_file(int fd, char *buf,int sock)
{
    int read_cnt;

    while((read_cnt = read(sock,buf,BUF_SIZE)) !=0)
        write(fd,buf,read_cnt);

    puts("Received File Data");
    write(sock, "Thank you", 10);
    close(fd);
}
```

file_server.c

```
#include "basics.h"


int main(int argc, char **argv)
{
        int serv_sock, clnt_sock, fd;
        char buf[BUF_SIZE] ={0};

        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,0644);//send file name
        serv_sock = init_sock();
        init_sock_addr(&serv_addr,sizeof(serv_addr),argv,0);
        post_sock(serv_sock, &serv_addr, sizeof(serv_addr));



        clnt_addr_size = sizeof(clnt_addr);
        clnt_sock = accept(serv_sock, (sap)&clnt_addr, &clnt_addr_size);

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

        write_clnt(fd,buf,clnt_sock);

    close(clnt_sock);
        close(serv_sock);

return 0;
}
```

---

file_client.c

```
#include "basics.h"


int main(int argc, char **argv)
{
        char buf[BUF_SIZE] = {0};
        int fd, sock, read_cnt;
        si serv_addr;

        if(argc !=3)
        {
                printf("use: %s <ip> <port>\n", argv[0]);
                exit(1);
        }

        fd = open("receive.txt" , O_CREAT | O_WRONLY);
        sock = init_sock();
        init_sock_addr(&serv_addr, sizeof(serv_addr), argv, 1);

        if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
                err_handler("connect() error");
        else
                puts("Connected .....");

        recv_file(fd,buf,sock);

        close(sock);
        return 0;
}
```

---

gcc -o serv file_server.c init.c write_clnt.c - server 의 컴파일

gcc -o clnt file_client.c init.c recv_file.c - client 의 컴파일

이렇게 파일을 분할시키면 필요한 기능만 쓸수있어 편리하다.

Web.c

```
#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	128

typedef struct sockaddr_in	si;
typedef struct sockaddr*	sp;


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

void send_error(FILE *fp){

	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> 오류 발생! 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/plain";
}

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){
		printf("Use: %s <port>\n", argv[0]);
		exit(1);
	}

	serv_sock = socket(PF_INET, SOCK_STREAM, 0);
	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)
		error_handler("bind() error");
	if(listen(serv_sock, 20) == -1)
		error_handler("listen() error");

	for(;;){
		clnt_addr_size = sizeof(clnt_addr);
		clnt_sock = accept(serv_sock, (sp)&clnt_addr,
&clnt_addr_size);
		printf("Connection Reques: %s:%d\n)"
```

```
                                        ,inet_ntoa(clnt_addr.sin_addr),
ntohs(clnt_addr.sin_port));

                pthread_create(&t_id, NULL, request_handler, &clnt_sock);
                pthread_detach(t_id);
        }

        close(serv_sock);
        return 0;
}
```

first.thml

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

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

</body>
</html>
```

아직 웹서버에대해서는 공부를 해야할거 같다.
근데 웹서버는 알고있어야할거같다.
나중에 프로잭트때 유용하게 쓰일거 같다.