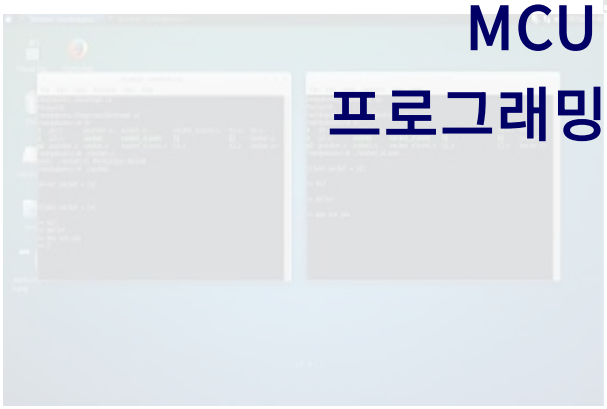

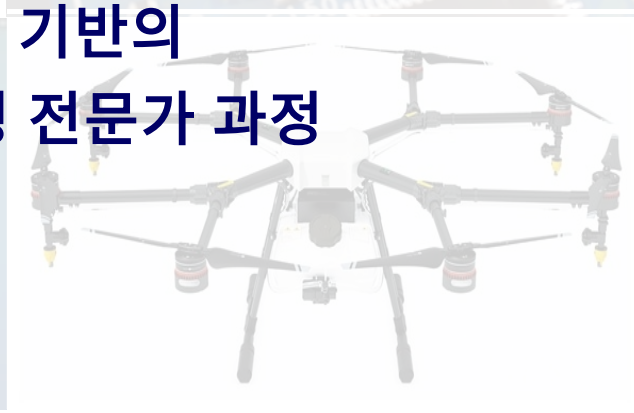

Xilinx Zynq FPGA,TI DSP, MCU 기반의 프로그래밍 전문가 과정

날 짜 : 2018 . 4. 3

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 정한별
hanbulkr@gmail.com

< 블록하기 > _(아직 세번 벙어리 시키고 말 못하게 하는거 구현 마무리.)

<blocking_server.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/time.h>
#include<sys/epoll.h>
#include<malloc.h>

#define BUF_SIZE                128
#define MAX_CLNT                256
#define SET_TIMES               6

typedef struct timeval    tv;
typedef struct sockaddr_in              si;
typedef struct sockaddr *               sp;

int cnt[MAX_CLNT];
int clnt_cnt = 0;
int idx;
int clnt_socks[MAX_CLNT];
int thread_pid[MAX_CLNT];
double runtime=0.0;
double load_ratio;
int flag;

pthread_mutex_t mtx;
tv start, end;


// 카운트를 위한 구조체 안씀.
typedef struct __count
{
        int send_client[0];
}count;


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

count *ct = 0;
// 시간을 구해주는 함수.
double get_runtime(tv start, tv end)
{
        end.tv_usec = end.tv_usec - start.tv_usec;
        end.tv_sec = end.tv_sec - start.tv_sec;
        end.tv_usec += end.tv_sec * 1000000;
```

```
	if((end.tv_usec / 1000000.0) > 10)
	{
		gettimeofday(&start, NULL);
		//cnt[]
	}

	//printf("runtime = %lf sec\n", end.tv_usec / 1000000.0);
	return end.tv_usec / 1000000.0;
}

void send_msg(char *msg, int len)
{
	int i;
	pthread_mutex_lock(&mtx);

	for(i = 0; i<clnt_cnt; i++)
		write(clnt_socks[i], msg, len);

	pthread_mutex_unlock(&mtx);
}

void proc_msg(char *msg, int len, int k)
{
	int i;
	//int cmp = atoi(msg);
	char smsg[256] = {0};
	char clnt_count[BUF_SIZE];

	pthread_mutex_lock(&mtx);

	cnt[k] += 1;

	sprintf(smsg,"[<말한 횟수 : %d>]\n",cnt[k]);
	write(clnt_socks[k], smsg, strlen(smsg));

	printf("cnt = %d\n", cnt[k]);

/*
	if(data[k] > cmp)
		sprintf(smsg, "greater than %d\n", cmp);
	else if(data[k] < cmp)
		sprintf(smsg, "less than %d\n", cmp);
	else
	{
		sprintf(clnt_count,"[<말한 횟수(1.6):%d>]\n",cnt[k]);
		write(clnt_socks[k], clnt_count, strlen(smsg));
	}
*/
	pthread_mutex_unlock(&mtx);
}


void *clnt_handler(void *arg)
{
	int clnt_sock = *((int *)arg);
	int str_len = 0, i;
```

```
char msg[BUF_SIZE];
char clnt_count[BUF_SIZE];
i = clnt_cnt - 1;

flag = 0;
//tv start, end;


pthread_mutex_lock(&mtx);
thread_pid[idx++] = getpid();
pthread_mutex_unlock(&mtx);

gettimeofday(&start, NULL);
while((str_len = read(clnt_sock , msg, sizeof(msg))) != 0){
        // 현재 말한 횟수 세는것은 한명만 된다.

        //ct->send_client[clnt_cnt-1] += 1;
        proc_msg(msg, str_len, i);
        send_msg(msg, str_len);

        // 끝나는 시간을 구함.
        gettimeofday(&end, NULL);

        //pthread_mutex_unlock(&mtx);

        runtime = get_runtime(start, end);
        if(runtime > 3)
        {
                gettimeofday(&start, NULL);
                cnt[i]=0;
        }
        load_ratio = cnt[i]/runtime;
        pthread_mutex_lock(&mtx);
        if(load_ratio > 5.7 || cnt[i]>15)
        {
                flag++;
                // 여기서 나갈때 같이 나가버림.
                if(flag == 3){
                        write(clnt_socks[i], "너 이제 진짜 말못함\n", 128);
                        shutdown(clnt_socks[i], SHUT_WR);
                        break ;
                }
                write(clnt_socks[i], "당신은 잠시 벙어리\n", 128);
                sleep(6);
                while(read(clnt_sock, msg, sizeof(msg)) > BUF_SIZE-1){
                        memset(msg, 0, sizeof(msg));
                }
                memset(msg, 0, sizeof(msg));
                gettimeofday(&start, NULL);
                write(clnt_socks[i], "당신은 기적으로 말할 수 있게 되었습니다. \n", 128);
                cnt[i]=0;

        }
        pthread_mutex_unlock(&mtx);

        printf("runtime = %lf sec\n", runtime);
        printf("{load_ratio} = %lf sec\n", load_ratio);
```

```
	}


	pthread_mutex_lock(&mtx);
	// 여기가 뭐하기 위한 건지 정확히 모르겠다.
	// 이제 알거 같다. 현재 클라이언트 뒤에 있는 놈을 종료되는 클라이언트위치에 옮겨 놓는 부분이다.
	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;

	unsigned int i, cnt = 0;


	ct = (count*)malloc(1024);

	gettimeofday(&start, NULL);

	for(i = 0; i < 777777777; i++)
		cnt++;

	gettimeofday(&end, NULL);

	// 여기서 시간을 구한다.
	runtime = get_runtime(start, end);


	if(argc != 2)
	{
		printf("Usage: %s <port>\n", argv[0]);
		exit(1);
	}

	pthread_mutex_init(&mtx, NULL);

	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 , 25)== -1)
                err_handler("listen() error!");

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


                pthread_mutex_lock(&mtx);
                ct->send_client[clnt_cnt] = 0;
                clnt_socks[clnt_cnt++] = clnt_sock;
                pthread_mutex_unlock(&mtx);

                printf("clnt_cnt (사람수) : %d\n", clnt_cnt);

                pthread_create(&t_id, NULL, clnt_handler, (void *) &clnt_sock);
                pthread_detach(t_id);
                printf("Connected Client IP: %s\n", inet_ntoa(clnt_addr.sin_addr));
        }
        close(serv_sock);
        return 0;
}
```

[헤더 파일 만들어서 모듈화 하기]

<server.h>

```
#ifndef __LOAD_TEST_H__

#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/time.h>
#include<sys/epoll.h>
#include<malloc.h>

typedef struct timeval   tv;
typedef struct sockaddr_in si;
typedef struct sockaddr *sap;
```

```
// err 체크를 하는 함수
void err_handler(char *msg);
//  서버에 기본적으로 쓰는 부분을 통으로 만듬, accept 전까지 만들었다.
void server_init(int argc , char **argv,int *serv_sock,int *clnt_sock, si *serv_addr,si *clnt_addr);
// 클라이언트가 메세지를 보낼 때를 위해서 만들었다.
void send_msg(char *msg, int len);
// 혹시 블록킹이나 타이밍을 재야 할 경우를 위해 만들었다.
double get_runtime(tv start, tv end);
// accept 부분을 위해 만들었다.
int accept_s(int serv_sock,si clnt_addr );
// socket 을 열은 fd 를 받아 오는 함수
int init_sock(void);
// 소켓에서 마지막 옵션으로 serv=0, clnt=1 일때 IPv4 설정 port, 이름등을 받는 함수.
void init_sock_addr(si *, int, char **, int);
// bind 와 listen 을 하는 함수
void post_sock(int, si *, int);


#endif
```

<server.c>

```
#include "server.h"

#define BUF_SIZE            128
#define MAX_CLNT            256
#define SET_TIMES           6

int cnt[MAX_CLNT];
int clnt_cnt = 0;
int idx;
int clnt_socks[MAX_CLNT];
int thread_pid[MAX_CLNT];
double runtime=0.0;
double load_ratio;
int flag;

pthread_mutex_t mtx;
tv start, end;

void err_handler(char *msg)
{
        fputs(msg, stderr);
        fputc('\n', stderr);
        exit(1);
}
double get_runtime(tv start, tv end)
{
        end.tv_usec = end.tv_usec - start.tv_usec;
        end.tv_sec = end.tv_sec - start.tv_sec;
```

```
	end.tv_usec += end.tv_sec * 1000000;

	if((end.tv_usec / 1000000.0) > 10)
	{
		gettimeofday(&start, NULL);
		//cnt[]
	}

	//printf("runtime = %lf sec\n", end.tv_usec / 1000000.0);
	return end.tv_usec / 1000000.0;
}

int init_sock(void)
{
	int sock;

	sock = socket(PF_INET, SOCK_STREAM, 0);

	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;

	if(opt)
	{
		serv_addr->sin_addr.s_addr = inet_addr(argv[1]);
		serv_addr->sin_port = htons(atoi(argv[2]));
	}
	else
	{
		serv_addr->sin_addr.s_addr = htonl(INADDR_ANY);
		serv_addr->sin_port = htons(atoi(argv[1]));
	}
}

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

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

```
}


int accept_s(int serv_sock,si clnt_addr)
{
	int clnt_sock;
	socklen_t clnt_addr_size;
	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() err");

	return clnt_sock;
}

void send_msg(char *msg, int len)
{
	int i;
	pthread_mutex_lock(&mtx);

	for(i = 0; i<clnt_cnt; i++)
		write(clnt_socks[i], msg, len);

	pthread_mutex_unlock(&mtx);
}


void server_init(int argc , char **argv,int *serv_sock,int *clnt_sock, si *serv_addr,si *clnt_addr)
{

	char msg[] = "Hello Network Programming";
	if(argc != 2)
	{
		printf("use: %s <port>\n", argv[0]);
		exit(1);
	}
	// 소켓은 파일이다, 원격에 있는 파일을 말하는 것. 그래서 ipc 를 쓴다.
	// 네트워크는 결국 원격의 ipc , 원격의 세마포어이다.
	// 소켓을 열어라 소크 스트림은  tcp 소켓을 사용한다는 뜻이다.
	// 리턴은 파일 디스크립터가 나온다.
	*serv_sock = socket(PF_INET, SOCK_STREAM, 0);

	if(*serv_sock == -1)
		err_handler("socket() error");

	// 서버 어드레스의  메모리를 한번 지워준다.
	memset(serv_addr, 0, sizeof(*serv_addr));
```

```
        serv_addr->sin_family =AF_INET; // 여기 패턴을 익혀야 한다.
        // 자신의 주소를 받겠다. 127.0.0.7 = 로컬호스트
        serv_addr->sin_addr.s_addr = htonl(INADDR_ANY);
        serv_addr->sin_port = htons(atoi(argv[1]));

        //스코프 바인딩. 서버의 ip 주소를 세팅한다. 127.0.0.7 이됨
        if(bind(*serv_sock, (sap)serv_addr, sizeof(*serv_addr)) == -1)
                err_handler("bind() error");

        // 20 명 까지 받겠다라는 뜻이다. 실제로 클라이언트  기다리는 곳이다.
        if(listen(*serv_sock, 20) == -1)
                err_handler("listen() error");


        // 밑으로 추가할 코드는 accept

        #if A
        for(;;)
        {
                addr_size = sizeof(*clnt_addr);
                clnt_sock = accept(serv_sock,(sap)&clnt_addr, &addr_size);


                pthread_mutex_lock(&mtx);
                ct->send_client[clnt_cnt] = 0;
                clnt_socks[clnt_cnt++] = clnt_sock;
                pthread_mutex_unlock(&mtx);

                printf("clnt_cnt (사람수) : %d\n", clnt_cnt);

                pthread_create(&t_id, NULL, clnt_handler, (void *) &clnt_sock);
                pthread_detach(t_id);
                printf("Connected Client IP: %s\n", inet_ntoa(clnt_addr.sin_addr));
        }
        close(serv_sock);

        #endif
}
// 블럭킹 서버 구현.
#if A
void *clnt_handler(void *arg)
{
        int clnt_sock = *((int *)arg);
        int str_len = 0, i;
        char msg[BUF_SIZE];
        char clnt_count[BUF_SIZE];
        i = clnt_cnt - 1;

        flag = 0;
```

```
//tv start, end;


pthread_mutex_lock(&mtx);
thread_pid[idx++] = getpid();
pthread_mutex_unlock(&mtx);

gettimeofday(&start, NULL);
while((str_len = read(clnt_sock , msg, sizeof(msg))) != 0){
        // 현재 말한 횟수 세는것은 한명만 된다.

        //ct->send_client[clnt_cnt-1] += 1;
        proc_msg(msg, str_len, i);
        send_msg(msg, str_len);

        // 끝나는 시간을 구함.
        gettimeofday(&end, NULL);

        //pthread_mutex_unlock(&mtx);

        runtime = get_runtime(start, end);
        if(runtime > 3)
        {
                gettimeofday(&start, NULL);
                cnt[i]=0;
        }
        load_ratio = cnt[i]/runtime;
        pthread_mutex_lock(&mtx);
        if(load_ratio > 5.7 || cnt[i]>15)
        {
                flag++;
                // 여기서 나갈때 같이 나가버림.
                if(flag == 3){
                        write(clnt_socks[i], "너 이제 진짜 말못함\n", 128);
                        shutdown(clnt_socks[i], SHUT_WR);
                        break ;
                }
                write(clnt_socks[i], "당신은 잠시 벙어리\n", 128);
                sleep(6);
                while(read(clnt_sock, msg, sizeof(msg)) > BUF_SIZE-1){
                        memset(msg, 0, sizeof(msg));
                }
                memset(msg, 0, sizeof(msg));
                gettimeofday(&start, NULL);
                write(clnt_socks[i], "당신은 기적으로 말할 수 있게 되었습니다. \n", 128);
                cnt[i]=0;

        }
        pthread_mutex_unlock(&mtx);
```

```
                printf("runtime = %lf sec\n", runtime);
                printf("{load_ratio} = %lf sec\n", load_ratio);

        }


        pthread_mutex_lock(&mtx);
        // 여기가 뭐하기 위한 건지 정확히 모르겠다.
        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;
}
#endif
```

<basic_server.c>

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

#define BUF_SIZE                128
#define MAX_CLNT                256
#define SET_TIMES               6


int cnt[MAX_CLNT];
//int clnt_cnt = 0;
int idx;
int clnt_socks[MAX_CLNT];
int thread_pid[MAX_CLNT];
//double runtime=0.0;
double load_ratio;
int flag;
```

```
pthread_mutex_t mtx;
tv start, end;

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";
	// 통으로 만들어둔 서버 intro 함수
	server_init(argc ,argv,&serv_sock,&clnt_sock,&serv_addr,&clnt_addr);
	// 이 억셉트를 for 에 넣을건지 안넣을지를 결정한다.
	clnt_sock = accept_s(serv_sock,clnt_addr);

	// accept 는 서버 소켓이 클라이언트의 실제 접속 허용을 해주는 곳이다.

	// 원격에 있는 클라이언트에게 롸이트를 한다.
	write(clnt_sock, msg, sizeof(msg));
	close(clnt_sock);
	close(serv_sock);

	return 0;
}
```

[Web 구현하기!!!]

<first.html>

```
 <!DOCTYPE html>
 <html>
  <head>
  <title>Hello daesung!!!</title>
  </head>
  <body>
  <h1>Hello Daesung!!!</h1>
  <iframe width="640" height="360" src="https://www.youtube.com/embed/ksLuVLIC    E7I"
frameborder="0" allow="autoplay; encrypted-media" allowfullscreen></ifr    ame>
  <p>This is a paragragh.</p>
 </body>
 </html>
```

<web_serv.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       100

typedef struct sockaddr_in      si;
typedef struct sockaddr*        sp;

void error_handling(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 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_handling("bind() error");
        if(listen(serv_sock, 20) == -1)
                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));
                pthread_create(&t_id, NULL, request_handler, &clnt_sock);
                pthread_detach(t_id);
        }

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

[비유 1]

→ A 씨는 프로그래머(우리) 사용자,
→ B 씨는 리눅스 운영체제, 그리고 Task_struct(context_switch),
→ C 씨는 데몬 ,C2 씨는 데몬 2.
→ (E, F, G)씨는 자식 프로세스들이 둔갑술을 하며 실행 하는 것.

[비유 2]

→ 카사노바는 cpu, 각각의 프로세스를 multi tasking 한다. 또한 context swiching 을 통해 병렬처리와 제어권을 넘겨준다.
→ 24 명의 여자는 24 개의 프로세스를 뜻한다.
→ cpu 는 오로지 한순간에 하나만 동작하기 때문에 여자 한명씩만난다는 비유가 된다.
→ 하나의 태스크만 실제 runnig 상태는 Run queue 에 들어와 있다라는 뜻이다.
→ 나머지 태스크들이 모두 대기중이 다는 뜻은 wait queue 에 있다는 뜻이다.
→ 새로 생성된 우선순위가 높은 태스크 실행 요청은 system call 이다. (interrupt)
→ task_struct 프로세스가 실행 될때 cpu 점유율을 결정해 놓는다. 그게 time_slice 다.

[비유 3]

→ 시그널 핸들러는 시그널이 발생할 시에 동작 수행을 하는 프로세스이다.
→ 시그널을 블록하는 녀석은 wait 이라는 함수이다.
→ 시그널은 정해진 상황에 대한 동작을 취해서 다른 통신 방식을 이용해야 더 많이 표현 할 수 있다.
→ 시그널은 이넘 형태로 저장 되어 있어 사실상 번호로 저장 되어 있다고 봐도 된다.

[비유 4]

→ tcp/ip 모드로 자신에게 연결되어 통신하는 녀석은 read 함수를 통해서(대기 상태로 있음) 패킷이 들어온다.
→ 김군은 소켓이다, 서버와 클라이언트의 연결후 상관이라는 내부 동작을 실행후 소켓을 종료했다.

[비유 5]

밑에 운영체제 동작과 다르지만 내생각은 이렇다.
→ 이군의 평소 물건을 창고에 넣어둘 때 방식은 queue, stack, 이진트리, avl 트리 등의 방식들을 의미한다.
→ 정리 방식: 자료구조의 종류
→ 창고안에 어떤 물건이냐는 처음 저장 하기로 생각한 것들,
→ 빈공간이 어디 있는가는 인덱스가 0 인 구조체를 이용했을 때 빈곳에 인덱스 위치.
→ 그 인덱스가(배열 1 개 ex_data[0] : 오이, data[1] : 가지) 가지는 공간이 저장한 물건들의 종류를 의미
→ 인덱스 별로 가지고 있는 값이 물건이 얼마나 있는가를 나타낸다.