

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



파일 I/O 제어,

프로세스 제어,

멀티 태스킹과 컨텍스트 스위칭,

signal 활용법,

IPC 기법

핵심철학 : 모든것은 파일이다.

System call : 유일한 소프트웨어 인터럽트, User 가 Kernel 에게 요청하는 작업을 의미한다.
open 은 숫자를 retrun 한다.

EX1- file_io2.c

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

#define ERROR -1

int main(void)
{
    int filedес;
    char pathname[]="temp.txt";
    if((filedes=open(pathname,O_CREAT | O_RDWR | 0644)) == ERROR)
// if((filedes=open(pathname,O_CREAT | O_RDWR ,0644)) == ERROR)
    {
        printf("File Open Error!\n");
        exit(1);
    }
    printf("fd=%d\n",filedes);

    close(filedes);

    return 0;
}
```

// 파일이 있다면 파일을 만들지마라.
// 파일을 읽고 쓸 수 있게 pathname 명으로 만들어라

EX2- file_io3.c

<pre>#include <fcntl.h> int main(void) { int filedес1,filedes2; filedес1 = open("data1.txt",O_WRONLY O_CREAT O_TRUNC, 0644); filedес2 = creat("data2.txt",0644); close(filedes1); close(filedes2); return 0; }</pre>	<p>//O_TRUNC : 무조건 한번 밀어버린다.(작업을 할때마다 새로 갱신함) 임시저장시 사용함.</p>
---	--

Flag 에 인자로 넘겨주는 값

- O_CREAT : 필요한 경우 파일을 생성한다.
- O_TRUNC : 존재하던 데이터를 모두 삭제한다.
- O_APPEND: 존재하던 데이터를 보존하고 뒤에 이어서 저장한다.
- O_RDONLY : 읽기 전용 모드로 파일을 연다
- O_WRONLY : 쓰기 전용 모드로 파일을 연다.
- O_RDWR : 읽기, 쓰기 겸용 모드로 파일을 연다.

EX3- file_io4.c 숫자값이 리턴된다.

```
#include <unistd.h>
#include <fcntl.h>

int main(void)
{
    int fdin,fdout;
    ssize_t nread;
    char buf[1024];
    fdin = open("temp1.txt",O_RDONLY);
    fdout = open("temp2.txt",O_WRONLY | O_CREAT | O_TRUNC , 0644);
    while((nread = read(fdin,buf, 1024))>0)
    {
        if(write(fdout, buf, nread) < nread)
        {
            close(fdin);
            close(fdout);
        }
    }
    close(fdin);
    close(fdout);
    return 0;
}
```

// 읽기전용, 읽을게 없으면 안열림.
// 실행할때마다 안에있는거 밀어버림.

// read(fd,buf,읽을크기) : fd 를 읽어와서 buf 에 읽을 크기만큼 집어넣음,
자기가 읽을 byte 크기 리턴.

// 읽을게 없으면 -값 리턴됨

// system call, read 로 fdin(인덱스)에 있는 값을 읽어와서 buf 에 집어넣
어줘

// write(fd,buf,홀 크기) : buf 에 있는 값이 nread 크기만큼 fd 에 써짐. 자
기가 쓸 byte 크기 리턴

// nread 만큼 썼으므로 nread 보다 작을 수 없음.

// 마찬가지로 system call 임.

→ cp 를 만듦. 파일복사..

files_struct → files → f_pos, : 위치를 저장 , 이중 포인터로 관리됨(즉 포인터배열이다.) // files *fd[] 배열의 인덱스
file_operations
inode → path → super_block

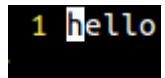
EX4- file_io5.c 파일의 용량 검사

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

int main(void)
{
    int filedес;
    off_t newpos;

    filedес = open("data1.txt",O_RDONLY);
    newpos = lseek(filedes, (off_t)0, SEEK_END); // 파일용량 검사
    printf("file size: %d\n",newpos);
}
```

data1.txt



```
1 hello
```

결과 : 

EX5- 파일 CP 코드 짜보기 → argc, argv

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

int main(int argc, char **argv)
{
    int i;
    int fdin,fdout;
    ssize_t nread;
    char buf[1024];

    if(argc!=3)
    {
        printf("인자 입력 3 개 하라고!\n");
        exit(-1);
    }
    for(i=0;i<argc;i++)
        printf("당신이 입력한 인자는 = %s\n",argv[i]);

    fdin = open(argv[1],O_RDONLY);
    fdout = open(argv[2],O_WRONLY | O_CREAT | O_TRUNC , 0644);

    while((nread = read(fdin,buf, 1024))>0)
    {
        if(write(fdout,buf,nread) < nread)
        {
            close(fdin);
            close(fdout);
        }
    }

    close(fdin);
```

argc : 프로그램을 실행할 때, 지정해 준 “명령행 옵션”의 “개수”가 저장되는 곳.

argv : 프로그램을 실행할 때, 지정해 준 “명령행 옵션의 문자열들”이 실제로 저장되는 배열
→ (**argv == *argv[])

<pre>close(fdout); return 0; }</pre>	
--------------------------------------	--

결과 :

```
koitt@koitt-Z20NH-AS51B5U:~/my_proj/Homework/sanghoonlee/lec/lhs/linux_system$ ./debug file_cp.c file_cp2.c
당신이 입력한 인자는 = ./debug
당신이 입력한 인자는 = file_cp.c
당신이 입력한 인자는 = file_cp2.c
koitt@koitt-Z20NH-AS51B5U:~/my_proj/Homework/sanghoonlee/lec/lhs/linux_system$ ls
data1.txt  debug      file_cp.c  file_io3.c  file_io5.c  temp2.txt  XDG_VTNR=7
data2.txt  file_cp2.c  file_io2.c  file_io4.c  temp1.txt   temp.txt
```


System call 을 사용하지 않는 것과 사용하는 것의 속도차이??

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

```
int main(void)
```

```
{
```

```
    FILE *fp = fopen("mycat.c","r");
```

```
    char buf[1024] = "\0";
```

```
    int ret;
```

```
    while(ret=fread(buf,1,sizeof(buf),fp))
```

```
    {
```

```
        usleep(1000000);
```

```
        fwrite(buf,1,ret,stdout);
```

```
    }
```

```
    fclose(fp); return 0;
```

```
}
```

```
// O_RDONLY 와 같은뜻. 시스템콜 아님.
```

```
//1byte 씩 1024 바이트를 읽어라.
```

```
//
```

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

```
int main(int argc, char **argv)
{
    int fd, ret;
    char buf[1024];
    if(argc != 2)
    {
        printf("Usage: mycat filename\n");
        exit(-1);
    }
    fd = open(argv[1], O_RDONLY);
    while(ret = read(fd, buf, sizeof(buf)))
    {
        write(1, buf, ret);
    }
    close(fd);
    return 0;
}
```

시스템 콜을 사용하면 더빠른 속도로 파일을 열고 닫고 쓸 수 있다!!

Read 시 키보드 지정해서 파일안에 쓰기

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

```
int main(void)
{
    int ret;
    char buf[1024]={0};
    int fd;
    fd = open("c.txt",O_CREAT|O_RDWR|0644);
    ret = read(0,buf,sizeof(buf));
    write(fd,buf,ret);

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

Read(0, ~,~) 키보드는 0 , 모니터는 1

Read 시 키보드 지정해서 파일안에 쓰기

```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include "my_scanf.h"
5
6 int main(void)
7 {
8     int nr;
9     char buf[1024]={0};
10
11     nr= my_scanf(buf,sizeof(buf));
12     printf("nr=%d\n",nr);
13     write(1,buf,nr);
14     return 0;
15 }
16
```

```
1 #include "my_scanf.h"
2
3 int my_scanf(char *buf, int size)
4 {
5     int nr = read(0,buf,size);
6     return nr;
7 }
```

write(1,~) : 모니터에 쓰기

```
1 #ifndef __MY_SCANF_H__ // 선언된 것이 있으면 이 헤더파일을 들어가지마라.
2 #define __MY_SCANF_H__ // 선언된 것이 없으면 my_scanf.h 가 0 으로 땀. (그냥상수임!)
3
4 #include <fcntl.h>
5 #include <unistd.h>
6
7 int my_scanf(char *, int);
8
9 #endif // 두번 선언하는 것을 방지한다.
10
11
12 // 분할되어있어 분간이쉬움
```

알고리즘 파트, 보드제어파트, 순수한 소프트웨어파트, 영상처리 파트가 있을 때, 한공간에 묶어 놓으면 안된다.
분할되어있어 분간하기 쉬움. (<> : 시스템헤더, “ “: 사용자정의(커스텀)헤더)

FILE 을 읽어서 라인,단어 갯수 확인.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

int main(int argc, char **argv)
{
    int fd= open(argv[1],O_RDONLY);
    int line =0;
    int word = 0;
    int flag=0;
    int cnt=0;
    char ch;
    if(argc!=2)
    {
        printf("You need1 mor parameter\n");
        printf("Usage:mywc filename \n");
        exit(-1);
    }

    if((fd=open(argv[1],O_RDONLY))<0) // 0 작으면 에러
    {
        perror("open()");
        exit(-1);
    }
```

숫자값 리턴, 0: 표준입력 1:표준출력 2:표준에러 3~5

```
while(read(fd,&ch,1))
{
    cnt++;
    if(ch == '\n')
        line++;
    if(ch!='\n'&& ch!='\t' &&ch !=' ')
    {
        if(flag==0)
        {
            word++;
            flag=1;
        }
    }
    else
    {
        flag=0;
    }
}
close(fd);
printf("%d %d %d %s\n",line,word,cnt,argv[1]);
return 0;
}
```

다운받기

:vs
:sp
:e 파일명

컨트롤 +w w

wget

<https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.4.tar.gz>

tar zxvf linux-4.4.tar.gz

/kernel/linux-

4.4/include/linux/sched.h → :set

hlsearch /구조체

드라이빙하기

```
vi ~/.vimrc
```

```
"ctags 설정"
```

```
set tags=/root/compiler/gcc-4.5.0/tags
```

```
(bp 이후에 set tags=/home/lhs/kernel/linux-4.4/tags 로바꿔준다)
```

```
if version >= 500
```

```
func! Sts()
```

```
    let st = expand("<word>")
```

```
    exe "sts ".st
```

```
endfunc
```

```
nmap ,st :call Sts()<cr>
```

```
func! Tj()
```

```
    let st = expand("<word>")
```

```
    exe "tj ".st
```

```
endfunc
```

```
nmap ,tj :call Tj()<cr>
```

```
endif
```

```
"cscope 설정"
```

```
set csprg=/usr/bin/cscope
```

```
set nocsverb
```

```
cs add /root/compiler/gcc-4.5.0/cscope.out
```

```
(bp 이후에 cs add /home/lhs/kernel/linux-4.4/cscope.out 로바꿔준다)
```

```
set cst=0
```

```
set cst
```

```
func! Cst()
```

```
    let cst = expand("<word>")
```

```
    new
```

```
    exe "cs find s ".cst
```

```
2. 명령어: vi ~/mkcscope.sh
```

```
#!/bin/sh
```

```
rm -rf cscope.files cscope.files
```

```
find . \( -name '*.c' -o -name '*.cpp' -o -name '*.cc' -o -name '*.h' -o -name  
 '*.S' \) -print > cscope.files
```

```
cscope -i cscope.files
```


<pre> if getline(1) == "" exe "q!" endif endfunc nmap ,css :call Csd()<cr> func! Csd() let csd = expand("<word>") new exe "cs find d ".csd if getline(1) == "" exe "q!" endif endfunc nmap ,csd :call Csd()<cr> func! Csg() let csg = expand("<word>") new exe "cs find g ".csg if getline(1) == "" exe "q!" endif endfunc nmap ,csg :call Csg()<cr> </pre>	<p>3. kernel 파일 → linux-4.4 파일 →</p> <p>명령어: sudo apt-get install ctags cscope 명령어 : ctags -R</p> <p>cd ~/ (홈으로)</p> <p>명령어: chmod 755 ~/mkcscope.sh → mkcscope.sh 가 초록색이 되야함. 명령어: sudo ~/mkcscope.sh /usr/local/bin/ 명령어 :ls /usr/local/bin/mkcscope.sh 명령어:mkcscope.sh → 빠져나오기 ctrl + D 명령어:vi -t task_struct ; 144 enter</p> <p>/files_struct 찾고 ctrl+] 누른다 1 누르고 enter</p> <p>struct file __rcu * fd_array[NR_OPEN_DEFAULT]; 위에 주석을 단다.</p> <p>/* open()을 통해서 얻게 되는 File Descriptor 의 번호는 결국 이 배열의 인덱스에 해당한다. 커널은 별도의 정보를 제공하지 않고 이 인덱스 정보만을 제공하므로 시스템 내부에 치명적인 손상을 줄 수 있는 포인터 주소등을 주지 않고도 유저가 파일을 제어할 수 있게 해줌 그래서 read, write, close 등에는 숫자만 전달하게됨 이 요청을 커널이 받으면 숫자값을 보고 어떤 파일을 제어해야 하는지 빠르게 파악할 수 있음. */</p>
	<p>152 include- 파일 구조체의 시작점. → q 누르고 152 를 친다. file_operations 에서 ctrl + } → 10</p> <p>빠져나올때는 ctrl + T path → 93 → struct dentry</p>

파일 묶어놓기

```
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

typedef struct
{
    char fname[20];
    int fsize;
}F_info;

int file_size(int fd)
{
    int fsize,old;
    old=lseek(fd,0,SEEK_CUR);
    fsize=lseek(fd,0,SEEK_END);
    lseek(fd,old,SEEK_SET);
    return fsize;
}
```

res.tar 2000

a.txt 700

a.txt	700
a.txt 의 내용	
b.txt	100
b.txt 의 내용	
c.txt	10
c.txt 의 내용	

```

int main(void)
{
    int src, dst, ret;
    char buf[1024];
    F_info info;
    int i;
    dst = open(argv[argc -1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    for(i=0;i<argc-2;i++)
    {
        src = open(argv[i+1],O_RDONLY);
        strcpy(info.fanme,argv[i+1]);
        info.fsize = file_size(src);
        write(dst,&info,sizeof(info));
        while(ret=read(src,buf,sizeof(buf)))
            write(dst,buf,ret);
        close(src);
    }
    close(dst);
    return 0;
}

```

file 이름 info.fanme 이라는 이름에 복사. → 구조체를 보자.

a.txt → hello

b.txt → linux system

c.txt → system call

./debug a.txt b.txt c.txt res.tar

xxd res.tar

파일 압축풀기

```
#include <fcntl.h>
typedef struct
{
    char fname[20];
    int fsize;
}F_info;
#define min(x,y)    (((x)<(y))? (x):(y))

int main(int argc, char *argv[])
{
    int src, dst, len, ret;
    F_info info;
    char buf[1024];
    src = open(argv[1], O_RDONLY);    //
    while(read(src,&info,sizeof(info))) // src
    {
        dst = open(info.fname,O_WRONLY|O_TRUNC|O_CREAT,0644); // a.txt 열어서 쓰기전용
        while(info.fsize >0)
        {
            len = min(sizeof(buf), info.fsize); // fsize 가 1024 를 넘으므로 최소값을 구함.
            ret = read(src,buf,len);
            write(dst,buf,ret);
            info.fsize -= ret;
        }
        close(dst);
    }
    close(src);
    return 0;
}
```

파일 삭제 후
./debug res

복사해와서 읽기

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

```
int main(void)
```

```
{
    char buf[1024];
    int fd[2];
```

```
    fd[0]=open("mytar.c", O_RDONLY);
    read(fd[0],buf,10);
    write(1,buf,10);
```

```
    fd[1] = open("mytar.c",O_RDONLY);
    read(fd[1],buf,10);
    write(1,buf,10);
    return 0;
```

```
}
```

실행 시킨 것 task_struct

open 을 하면 별도의 타입 디스크립트가 생긴다.

Quiz1

임의의 난수를 10 개 발생시켜서 이 값을 배열에 저장하고, 배열에 저장된 값을 파일에 기록한다. (중복안됨.)
그리고 이 값을 읽어서 Queue 를 만든다.
이후에 여기 저장된 값 중 짝수만 선별하여 모두 더한 후에 더한 값을 파일에 저장하고, 저장한 파일을 읽어 저장된 값을 출력하도록한다.

-me

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

int* randomm(int arr[])
{
    int i,k;
    for(i=0;i<10;i++)
    {
        arr[i]=rand()%10+1;
        for(k=0;k<i;k++)
        {
            while(arr[i]==arr[k])
            {
                arr[i]=rand()%10+1;
                k=0;
            }
        }
    }
    return arr;
}
```

```
int main(void)
{
    int arr[10];
    int* arr_m=randomm(arr);
    int i;
    int fd_array;
    int buf[1024]={0};

    for(i=0;i<10;i++)
        printf("%d\n",arr_m[i]);

    fd_array = open("fd_array.txt",O_WRONLY | O_CREAT | O_TRUNC, 0644);

    for(i=0;i<10;i++)
    {
        sprintf(buf,"%d\n",arr_m[i]);
        // printf("buf=%s\n",buf);

        write(fd_array,buf,strlen(buf));
    }
    close(fd_array);

    read(
    return 0;
}
```

-teacher

1

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#include <unistd.h>
#include <fcntl.h>

int extract_idx;

typedef struct __queue
{
    int data;
    struct __queue *link;
} queue;

bool is_dup(int *arr, int cur_idx)
{
    int i, tmp = arr[cur_idx];

    for(i = 0; i < cur_idx; i++)
        if(tmp == arr[i])
            return true;

    return false;
}
```

2

```
void init_rand_arr(int *arr, int size)
{
    int i;

    for(i = 0; i < size; i++)
    {
redo:
        arr[i] = rand() % 10 + 1;

        if(is_dup(arr, i))
        {
            printf("%d dup! redo rand()\n", arr[i]);
            goto redo;
        }
    }
}

void print_arr(int *arr, int size)
{
    int i;

    for(i = 0; i < size; i++)
        printf("arr[%d] = %d\n", i, arr[i]);
}

queue *get_queue_node(void)
{
    queue *tmp;
    tmp = (queue *)malloc(sizeof(queue));
    tmp->link = NULL;
    return tmp;
}
```


3

```
void enqueue(queue **head, int data)
{
    if(*head == NULL)
    {
        *head = get_queue_node();
        (*head)->data = data;
        return;
    }

    enqueue(&(*head)->link, data);
}

void extract_even(queue *head, int *extract)
{
    queue *tmp = head;

    while(tmp)
    {
        if(!(tmp->data % 2))
            extract[extract_idx++] = tmp->data;
        tmp = tmp->link;
    }
}
```

4

```
int main(void)
{
    int i, fd, len, sum = 0;
    char *convert[10] = {0};
    int arr[11] = {0};
    char tmp[32] = {0};
    int extract[11] = {0};
    int size = sizeof(arr) / sizeof(int) - 1;
    queue *head = NULL;

    srand(time(NULL));

    init_rand_arr(arr, size);
    print_arr(arr, size);

    for(i = 0; i < size; i++)
        enqueue(&head, arr[i]);

    extract_even(head, extract);
    printf("\nExtract:\n");
    print_arr(extract, extract_idx);

    fd = open("log.txt", O_CREAT | O_WRONLY |
O_TRUNC, 0644);

    for(i = 0; i < extract_idx; i++)
        sum += extract[i];

    sprintf(tmp, "%d", sum);
    write(fd, tmp, strlen(tmp));
    close(fd);
}
```

```
#if 0
    for(i = 0; i < extract_idx; i++)
    {
        int len;
        char tmp[32] = {0};

        sprintf(tmp, "%d", extract[i]);
        len = strlen(tmp);
        convert[i] = (char *)malloc(len + 1);
        strcpy(convert[i], tmp);
        printf("tmp = %s\n", tmp);
    }
#endif

    return 0;
}
```

Quiz2

카페에 있는 50 번 문제(성적 관리 프로그램)을 개조한다.
어떻게 개조할 것인가 ?
기존에는 입력 받고 저장한 정보가 프로그램이 종료되면 날아갔다.
입력한 정보를 영구히 유지할 수 있는 방식으로 만들면 더 좋지 않을까 ?

* 조건

1. 파일을 읽어서 이름 정보와 성적 정보를 가져온다.
2. 초기 구동시 파일이 없을 수 있는데
이런 경우엔 읽어서 가져올 정보가 없다.
3. 학생 이름과 성적을 입력할 수 있도록 한다.
4. 입력된 이름과 성적은 파일에 저장되어야 한다.
5. 당연히 통계 관리도 되어야한다(평균, 표준 편차)
6. 프로그램을 종료하고 다시 키면
파일에서 앞서 만든 정보들을 읽어와서 내용을 출력해줘야 한다.
7. 언제든지 원하면 내용을 출력할 수 있는 출력함수를 만든다.
[특정 버튼을 입력하면 출력이 되게 만듦]
(역시 System Call 기반으로 구현하도록 함)

```

#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

typedef struct __queue
{
    int score;
    char *name;
    struct __queue *link;
} queue;

void disp_student_manager(int *score, char *name, int size)
{
    char *str1 = "학생 이름을 입력하시오: ";
    char *str2 = "학생 성적을 입력하시오: ";
    char tmp[32] = {0};           // read 시에 char 형을 입력해야하지만 int 형이기에 설정해줌

    write(1, str1, strlen(str1)); // 1 표준입력
    read(0, name, size);
    write(1, str2, strlen(str2));
    read(0, tmp, sizeof(tmp));

    *score = atoi(tmp);           // tmp 값을 char->int 로 변환시켜서 score 에 넣는다 반대는 sprintf(int->char)
}

void confirm_info(char *name, int score)
{
    printf("학생 이름 = %s\n", name);
    printf("학생 성적 = %d\n", score);
}

```

```
queue *get_queue_node(void)
{
    queue *tmp;

    tmp = (queue *)malloc(sizeof(queue));
    tmp->name = NULL;
    tmp->link = NULL;

    return tmp;
}

void enqueue(queue **head, char *name, int score)
{
    if(*head == NULL)
    {
        int len = strlen(name);
        (*head) = get_queue_node();
        (*head)->score = score;
        (*head)->name = (char *)malloc(len + 1);
        strncpy((*head)->name, name, len);
        return;
    }
    enqueue(&(*head)->link, name, score);
}

void print_queue(queue *head)
{
    queue *tmp = head;
    while(tmp)
    {
        printf("name = %s, score = %d\n", tmp->name, tmp->score);
        tmp = tmp->link;
    }
}
```

```

void remove_enter(char *name)
{
    int i;

    for(i = 0; name[i]; i++)
        if(name[i] == '\n')
            name[i] = '\0';
}

int main(void)
{
    int cur_len, fd, btn = 0;
    int score;

    // Slab 할당자가 32 byte 를 관리하기 때문에 성능이 빠름
    char name[32] = {0};
    char str_score[32] = {0};
    char buf[64] = {0};

    queue *head = NULL;

    for(;;)
    {
        printf("1 번: 성적 입력, 2 번: 파일 저장, 3 번: 파일 읽기, 4 번: 종료\n");
        scanf("%d", &btn);

        switch(btn)
        {
            case 1:
                disp_student_manager(&score, name, sizeof(name));
                remove_enter(name);
                confirm_info(name, score);

                enqueue(&head, name, score);

```

```
    print_queue(head);
    break;
case 2:
    // 만약 파일 없다면 생성
    // 있다면 불러서 추가
    if((fd = open("score.txt", O_CREAT | O_EXCL | O_WRONLY, 0644)) < 0)
        fd = open("score.txt", O_RDWR | O_APPEND);

    /* 어떤 형식으로 이름과 성적을 저장할 것인가 ?
       저장 포맷: 이름,성적\n */
    strncpy(buf, name, strlen(name));
    cur_len = strlen(buf);
    //printf("cur_len = %d\n", cur_len);
    buf[cur_len] = ',';
    sprintf(str_score, "%d", score);
    strncpy(&buf[cur_len + 1], str_score, strlen(str_score));
    buf[strlen(buf)] = '\n';
    //printf("buf = %s, buf_len = %lu\n", buf, strlen(buf));

    write(fd, buf, strlen(buf));

    close(fd);

    break;
```

```

case 3:
    if((fd = open("score.txt", O_RDONLY)) > 0)
    {
        int i, backup = 0;
        // 이름 1, 성적 1\n
        // .....
        // 이름 n, 성적 n\n
        read(fd, buf, sizeof(buf));

        for(i = 0; buf[i]; i++)
        {
            if(!(strcmp(&buf[i], ",", 1)))
            {
                strcpy(name, &buf[backup], i - backup);
                backup = i + 1;
            }
            if(!(strcmp(&buf[i], "\n", 1)))
            {
                strcpy(str_score, &buf[backup], i - backup);
                backup = i + 1;
                enqueue(&head, name, atoi(str_score));
            }
        }
        print_queue(head);
    }
    else
        break;

    break;

case 4:
    goto finish;
    break;

```



```
        default:
            printf("1, 2, 3, 4 중 하나 입력하셈\n");
            break;
    }
}

finish:
    return 0;
}
```

1. dup

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

int main(void)
{
    int fd;
    fd = open("a.txt",O_WRONLY|O_CREAT|O_TRUNC,0644);
    close(1);
    dup(fd);
    printf("출력될까?\n");
    return 0;
}
```

모니터로 출력
dup → close 된놈을 fd 가 대체함.
Printf 의 역할은 1(즉,모니터) 의 역할을 한다.
그러나 close(1)을 하면서 printf 의 역할을 fd 가 가져감.

2. cat 명령 뒤에 파일이름을 입력하면 그 파일의 내용을 출력한다.

```
cat file_out.c > ccc; // 리다이렉션 기호(>)를 사용하여 입력한 내용으로 새로운 파일을 만듭니다
cat < ccc;
cat ccc; // 파일 내용 출력;
cat < ccc > ddd;
```

3 file_out3.c

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

int main(void)
{
    int fd;
    char buff[1024];
    fd = open("a.txt",O_RDONLY);
    close(0);
    dup(fd);
    gets(buff);
    printf("출력될까 ?\n");
    // printf("%s",buff);
    return 0;
}
```

gets 는 입력받는 값을 저장한다. 원래는 화면에서 입력받게되는것
close(0)으로 gets() 가 씹힌다.

→ buff

파일자체인 fd 가 dup 함수를 통해 0 번으로 대체되니까
gets 함수를 통해 buff 입력자체가 파일이된다!

Ps 명령어는 process 의 state 를 보는 명령어에 해당한다.

리눅스 명령어 ps, tall

1. Ps -ef | // 지금 돌아가고있는 프로세스들을 키는것

// PID : 프로세스가 누군지 알수 있다.

```
lhs@lhs-NH:~/my_proj/github/Homework/sanghoonlee/lec/lhs/linux_system/3_21$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  0 09:20 ?           00:00:01 /sbin/init splash
root           2        0  0 09:20 ?           00:00:00 [kthreadd]
root           4        2  0 09:20 ?           00:00:00 [kworker/0:0H]
root           6        2  0 09:20 ?           00:00:00 [mm_percpu_wq]
root           7        2  0 09:20 ?           00:00:00 [ksoftirqd/0]
root           8        2  0 09:20 ?           00:00:02 [rcu_sched]
root           9        2  0 09:20 ?           00:00:00 [rcu_bh]
root          10        2  0 09:20 ?           00:00:00 [migration/0]
root          11        2  0 09:20 ?           00:00:00 [watchdog/0]
root          12        2  0 09:20 ?           00:00:00 [cpuhp/0]
root          13        2  0 09:20 ?           00:00:00 [cpuhp/1]
root          14        2  0 09:20 ?           00:00:00 [watchdog/1]
root          15        2  0 09:20 ?           00:00:00 [migration/1]
root          16        2  0 09:20 ?           00:00:00 [ksoftirqd/1]
root          18        2  0 09:20 ?           00:00:00 [kworker/1:0H]
root          19        2  0 09:20 ?           00:00:00 [cpuhp/2]
root          20        2  0 09:20 ?           00:00:00 [watchdog/2]
root          21        2  0 09:20 ?           00:00:00 [migration/2]
root          22        2  0 09:20 ?           00:00:00 [ksoftirqd/2]
root          24        2  0 09:20 ?           00:00:00 [kworker/2:0H]
root          25        2  0 09:20 ?           00:00:00 [cpuhp/3]
root          26        2  0 09:20 ?           00:00:00 [watchdog/3]
root          27        2  0 09:20 ?           00:00:00 [migration/3]
root          28        2  0 09:20 ?           00:00:00 [ksoftirqd/3]
root          30        2  0 09:20 ?           00:00:00 [kworker/3:0H]
root          31        2  0 09:20 ?           00:00:00 [kdevtmpfs]
root          32        2  0 09:20 ?           00:00:00 [netns]
root          36        2  0 09:20 ?           00:00:00 [khungtaskd]
root          37        2  0 09:20 ?           00:00:00 [oom_reaper]
root          38        2  0 09:20 ?           00:00:00 [writeback]
```

1.1 ps -ef | grep bash // grep 으로 bash 를 찾는다.

(ps -ef 창)

```
lhs      4921   1416   0 10:04 ?        00:00:26 /usr/lib/gnome-terminal/gnome
lhs      4928   4921   0 10:04 pts/0    00:00:00 bash
lhs      5649   4928   0 10:56 pts/0    00:00:00 vi quiz2.c
lhs      5650   5649   0 10:56 ?        00:00:00 /usr/bin/cscope -dl -f /home/
lhs      5652   4921   0 10:56 pts/2    00:00:00 bash
lhs      6545   5652   0 11:54 pts/2    00:00:00 vi quiz1_2.c
lhs      6546   6545   0 11:54 ?        00:00:00 /usr/bin/cscope -dl -f /home/
lhs      6549   4921   0 11:54 pts/19   00:00:00 bash
root     6713      2   0 12:08 ?        00:00:03 [kworker/0:1]
root     6852      2   0 12:45 ?        00:00:04 [kworker/2:2]
root     6898      2   0 13:01 ?        00:00:00 [kworker/u16:2]
lhs      6908   1416   3 13:02 ?        00:01:19 /opt/google/chrome/chrome
lhs      6914   6908   0 13:02 ?        00:00:00 cat
```

(Ps -ef | grep bash 창)

```
lhs@lhs-NH:~/my_proj/github/Homework/sanghoonlee/lec/lhs/linux_system/3_21$ ps -ef|grep bash
lhs      4928   4921   0 10:04 pts/0    00:00:00 bash
lhs      5652   4921   0 10:56 pts/2    00:00:00 bash
lhs      6549   4921   0 11:54 pts/19   00:00:00 bash
lhs      7700   6549   0 13:33 pts/19   00:00:00 grep --color=auto bash
```

1.2 ps -ef | grep bash | grep -v grep // 찾는 grep 을 제외

```
lhs@lhs-NH:~/my_proj/github/Homework/sanghoonlee/lec/lhs/linux_system/3_21$ ps -ef|grep bash|grep -v grep
lhs      4928   4921   0 10:04 pts/0    00:00:00 bash
lhs      5652   4921   0 10:56 pts/2    00:00:00 bash
lhs      6549   4921   0 11:54 pts/19   00:00:00 bash
```

1.3 ps -ef | grep bash | grep -v grep | awk '{print \$2}'

(ps -ef 창)

```
lhs@lhs-NH:~/my_proj/github/Homework/sanghoonlee/lec/lhs/linux_system/3_21$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  0  09:20 ?           00:00:01 /sbin/init splash
root           2        0  0  09:20 ?           00:00:00 [kthreadd]
root           4        2  0  09:20 ?           00:00:00 [kworker/0:0H]
root           6        2  0  09:20 ?           00:00:00 [mm_percpu_wq]
root           7        2  0  09:20 ?           00:00:00 [ksoftirqd/0]
root           8        2  0  09:20 ?           00:00:02 [rcu_sched]
root           9        2  0  09:20 ?           00:00:00 [rcu_bh]
root          10        2  0  09:20 ?           00:00:00 [migration/0]
root          11        2  0  09:20 ?           00:00:00 [watchdog/0]
root          12        2  0  09:20 ?           00:00:00 [cpuhp/0]
root          13        2  0  09:20 ?           00:00:00 [cpuhp/1]
root          14        2  0  09:20 ?           00:00:00 [watchdog/1]
root          15        2  0  09:20 ?           00:00:00 [migration/1]
root          16        2  0  09:20 ?           00:00:00 [ksoftirqd/1]
root          18        2  0  09:20 ?           00:00:00 [kworker/1:0H]
root          19        2  0  09:20 ?           00:00:00 [cpuhp/2]
root          20        2  0  09:20 ?           00:00:00 [watchdog/2]
root          21        2  0  09:20 ?           00:00:00 [migration/2]
root          22        2  0  09:20 ?           00:00:00 [ksoftirqd/2]
root          24        2  0  09:20 ?           00:00:00 [kworker/2:0H]
root          25        2  0  09:20 ?           00:00:00 [cpuhp/3]
root          26        2  0  09:20 ?           00:00:00 [watchdog/3]
root          27        2  0  09:20 ?           00:00:00 [migration/3]
root          28        2  0  09:20 ?           00:00:00 [ksoftirqd/3]
root          30        2  0  09:20 ?           00:00:00 [kworker/3:0H]
root          31        2  0  09:20 ?           00:00:00 [kdevtmpfs]
root          32        2  0  09:20 ?           00:00:00 [netns]
root          36        2  0  09:20 ?           00:00:00 [khungtaskd]
root          37        2  0  09:20 ?           00:00:00 [oom_reaper]
root          38        2  0  09:20 ?           00:00:00 [writeback]
```

(ps -ef 창) 의 UID PID PPID C 번호를 찾아준다.

```
lhs@lhs-NH:~/my_proj/github/Homework/sanghoonlee/lec/lhs/linux_system/3_21$ ps -ef | grep bash | grep -v grep | awk '{print $2}'
4928
5652
6549
lhs@lhs-NH:~/my_proj/github/Homework/sanghoonlee/lec/lhs/linux_system/3_21$ ps -ef
```

2. tail 명령어는 파일 내용의 마지막부터 읽을 때 주로 사용한다.

tail -c 20 mytar.c - 문자수 (끝에서부터)

tail -n 1 mytar.c - 라인수 (끝에서부터)

1. (intel cpu) 경우에도 512 byte 내에서 부트로드를 동작한다.

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
    int i;
    char ch = 'a';
    int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    lseek(fd, 512-1, SEEK_SET);
    write(fd, &ch, 1);
    close(fd);
    return 0;
}
```

// 파일의 시작에서 512-1 번 건너뛴다.
// 다음에 써야 할 내용은 512 번째가 된다. 그곳에 a 를 쓴 것.

실행

```
./debug mbr.txt // master code number
xxd mbr.txt     // xxd <파일명> : shell 상에서 binary 파일의 hexdump 를 보여주는 명령어/
```


프로세스는 CPU의 추상화

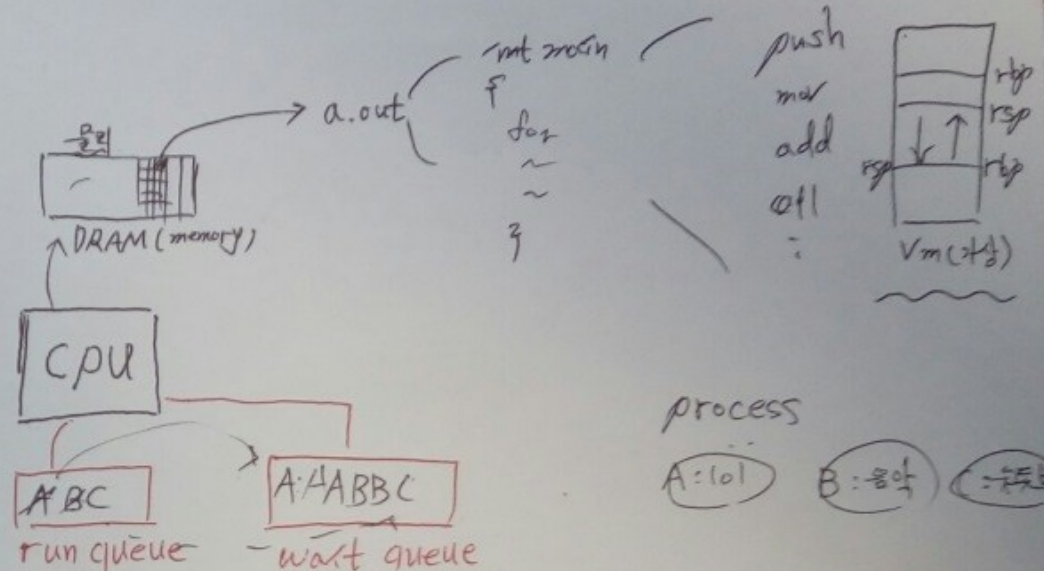
2. 프로세스가 여러개 있는데
어떻게 전부 동시에 실행되는지.

- 1. 정말 동시에일까? → 싱글코어
- 2. 병렬 처리는 되지? → 멀티코어.

→ CPU는 오로지 한 순간에 한가지 연산만 수행한다.

⇒ Context switching → 멀티태스킹 가능.
프로세스 하나당 task_struct가 만들어진다.

⇐



⇒ 프로세스가 CPU와 범용레지이 접근해서 우선순위가 높은 순서로 동작한다.

프로세스는 CPU의 동작 제어권을 가질 때마다 증가하게 된다.

서로 가져가려고 하는 모습을 보임. 병렬적으로 프로세스가 멀티태스킹되는 것처럼 보인다. 멀티코어면??

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
```

```
int main(void)
```

수행하고 다른 terminal 에서 cat > myfifo 를 수행함

```
{
    int fd,ret;
    char buf[1024];
    mkfifo("myfifo");           // 특수 파일 만들기
    fd=open("myfifo",O_RDWR);   // 읽기만
    for(;;)
    {
        ret = read(0,buf,sizeof(buf)); // (blocking) 키보드입력 받을게. 받을 때까지 다음 작업 없어~ 자기가 읽을 값 byte 리턴
        buf[ret-1]=0;             // ret-1 을 하는 이유??
        printf("Keyboard input: [%s]\n",buf); // 키보드에 쓴값 출력해~
        read(fd,buf,sizeof(buf)); // blocking 이번엔 다른 터미널에서 myfifo 파일에 쓴 값 읽어 올거야
        buf[ret -1] = 0;
        printf("Pipe input :[%s]\n",buf); // 하드웨어가 block 을 벗어나게해줌
    }
    return 0;
}
```

Blocking nonblocking - 제어권

read 라는 시스템콜은 블로킹 함수이다.
블로킹은 입력을 할때까지 제어권을 넘겨주지 않겠다는 것.

블록킹이 좋냐 논블록킹이 좋냐?

아주 빠르게 통신해야할 때는 논블록킹이 좋다.

반드시 순차적으로 진행되어야하는 것은 블록킹이 좋다.

d - directory

c - character device

b - block device

p 는 pipe

ls -al

블록은 특정단위에따라 움직이고, character 는 순서

물리메모리의 최소 단위는 4k byte 즉, 하드디스크도 4k byte 로 움직임 .. 즉 블록 dram 도 블록 ,

실행 방법 : 파일 컴파일: a.out → 만든 터미널창에서 mkfifo myfifo 를 통해 myfifo 를 만든다. 그리고 다른 터미널창을 띄워 cat > myfifo 를 통해 myfifo 에 들어간다. 그리고 블라블라 쓰면 블라블라가 나온다.

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    int fd,ret;
    char buf[1024];
    fd = open("myfifo",O_RDWR);
    fcntl(0,F_SETFL,O_NONBLOCK); // 0 을 nonblock 으로 설정한다.
    fcntl(fd,F_SETFL,O_NONBLOCK); // fd 를 nonblock 으로 설정한다.
    for(;;)
    {
        if((ret=read(0,buf,sizeof(buf))) > 0)
        {
            buf[ret-1]=0;
            printf("Keyboard input : %s\n",buf);
        }
        if((ret =read(fd,buf,sizeof(buf)))>0)
        {
            buf[ret-1]=0;
            printf("Pipe input:[%s]\n",buf);
        }
    }
    close(fd);
    return 0;
}
```

읽을 것이 있다면 처리하고 없다면 제어권을 넘겨준다.

○ 함수원형

```
#include <fcntl.h>

int fcntl(int fd, int cmd, int arg);
```

- fd : 제어 할 파일의 파일 기술자
- cmd : 파일 기술자에 대한 특성을 제어하기 위한 매개변수
- arg : cmd에 의해 결정되는 선택적(optional)인 값

○ 반환 값

- 성공시 : cmd에 따라 다른 값을 가진다.
- 실패시 : -1

(2) cmd 매개변수

○ F_DUPED

: 기존 파일 기술자를 복사하기 위해 사용된다. arg에 복제할 fd 값을 넘겨주고, 성공시 복제한 새로운 fd를 반환한다. arg로 입력받은 fd 값이 있다면, arg보다 큰 값 중 가장 작은 값으로 복제가 된다. (이 점이 dup2()함수와 다르다. dup2()함수는 지정한 fd로만 복제가 된다.)

○ F_GETFD

: 기존 파일 기술자의 flags를 조회하기 위해 사용된다. 현재는 FD_CLOEXEC 플래그 하나만 반환한다. FD_CLOEXEC 플래그는 하나의 프로세스에서 새로운 프로세스를 실행시킬 때 열려있는 fd 를 그대로 넘겨준다. 새로운 프로세스에 열린 fd 값을 상속시키는 것이 디폴트로 FD_CLOEXEC 플래그를 해제하는 것이고, 0 으로 사용할 수도 있다. 반대로 새로운 프로세스에 열린 fd 값을 상속시키지 않게 하는 것은 FD_CLOEXEC 플래그를 설정하는 것이고, 1로 사용할 수도 있다.

○ F_SETFD

: fd 에 FD_CLOEXEC 플래그를 설정할때 사용된다. 설정할 새로운 플래그 값은 세번째 인수 arg에 지정한다.

○ F_GETFL

: 파일 상태 flags 를 조회할때 사용된다. open() 함수 호출 시 설정한 플래그 값들을 반환해준다. open() 함수 관련 포스팅에서 알아봤었지만 O_RDONLY, O_WRONLY, O_RDWR 는 서로 상호배타적으로 하나의 파일에서는 이 중 하나의 플래그만 가질 수 있다. 따라서 O_ACCMODE 를 이용해서 어떤 플래그를 가지고 있는지 확인해야 한다.

○ F_SETFL

: 파일 상태 flags 를 설정할때 사용된다. 세 번째 인수 arg에 파일 상태 플래그를 설정한다. 변경할 수 있는 플래그들은 O_APPEND, O_NONBLOCK, O_SYNC, O_DSYNC, O_RSYNC, O_FSYNC, O_ASYNC 뿐이다.

