

**Xilinx Zynq FPGA, TI DSP, MCU 기반의  
프로그래밍 및 회로 설계 전문가 과정  
#20**

**강사 : Innova Lee(이 상훈)**

**학생 : 김 시윤**

## 1.배운내용 복습.

### QUIZ 2번

#### ---헤더---

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

#### ---Queue 구조체---

```
typedef struct __queue
{
    int score;
    char *name;
    struct __queue *link;
} queue;
//원래는 큐로 안하고 RB트리로 관리한다고함.
```

#### ---Student manager---

```
void disp_student_manager(int *score, char *name, int size)
{
    char *str1 = "학생 이름을 입력하시오: ";
    char *str2 = "학생 성적을 입력하시오: ";
    //str에 print할 문자열이 적혀있다
    char tmp[32] = {0};
```

```
    write(1, str1, strlen(str1)); //str1을 모니터에 print 해준다
    read(0, name, size);
    //키보드에 입력된걸 name이라는 캐릭터 포인터에 저장한다
    write(1, str2, strlen(str2));
    //str2를 모니터에 프린트한다
    read(0, tmp, sizeof(tmp));
    //키보드에서 입력된 문자열을 tmp에 저장한다.
    *score = atoi(tmp);
    //여기서 tmp는 점수 이기 때문에 숫자이다. 우리가 점수를 인트형으로 선언해
    저장했기 때문에 tmp를 atoi를 통해 인트형으로 변환해 반환해준다.
}
```

#### ---확인 함수---

```
void confirm_info(char *name, int score)
{
    printf("학생 이름 = %s\n", name);
    printf("학생 성적 = %d\n", score);
}
//name 과 score 에 값이 잘 들어있나 확인해주는 함수
```

#### ---Queue---

```
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);  
} //queue에 저장되어 관리되는게 학생 이름과 성적이라 데이터 변수가 두 개다.

```
void print_queue(queue *head)
{
    queue *tmp = head;

    while(tmp)
    {
        printf("name    = %s,    score    = %d\n",    tmp->name,
tmp->score);

        tmp = tmp->link;
    }
}
```

### ---ENTER 지우는 함수---

```
void remove_enter(char *name)
{
    int i;

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

```
        name[i] = '\0';
}
```

//name 이라는 배열에 I값이 엔터면 NULL문자로 변환해라.

/\* 수업시간에 strlen을 사용하여 write를 -1만큼 해도 되냐고 물었더니 된다고 했다. \*/

### ---main---

```
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(;;)
    {
        //btn 은 사용자가 사용할 메뉴를 받을 변수
        printf("1 번: 성적 입력, 2 번: 파일 저장, 3 번: 파일 읽기, 4 번: 종료\n");
        scanf("%d", &btn);

        switch(btn)
        {
            case 1: //btn==1일 때 (성적입력)
                disp_student_manager(&score, name, sizeof(name));

                //display 해주는 함수로 가서 동작을한다. 학생이름을 입력하세요.가
                write 돼서 모니터에 나오고 입력한 이름은 name에 문자열로 저장되어 반환된다.
                score는 tmp라는 문자열에 저장되고 tmp는 atoi를 통해 int형으로 변환되어 반환한다.
                sizeof name 은 name의 길이 만큼 읽기 위해서이다. 설정안해주면 다른 이상한 외계어도
                나와버린다.

                remove_enter(name);

                //이름이 저장된 name의 배열을 읽어 ENTER를 찾아 NULL문자로 바꾸는 함수이다.
                name이라는 배열에 ENTER가 저장되는 이유는 이름을 입력하고 엔터를 누르기 때문이다.
```

이를 없애는 방법은 strlen(name-1); 이라는 방법이 있다.

```
confirm_info(name, score);
```

//입력된 값이 잘 들어왔나 확인해 주는 함수이다.

```
enqueue(&head, name, score);
```

//입력받아 저장한 값들을 큐에 다시 저장한다.

```
print_queue(head);
```

//큐를 출력한다 입력했던 결과가 저장되어 나온다.

```
break;
```

```
case 2: //키보드에 입력한 숫자가 2일 때 파일생성기능
```

```
// 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);
```

//fd를 두 번 오픈한다. 저번에 퀴즈1번 문제를 답을 안보고 직접 코딩했을 때 오픈을 두 번했었는데, 시스템 콜을 두 번했을경우 성능 또는 파이프라인등이 어떻게 되는지 궁금하다.

```
/* 어떤 형식으로 이름과 성적을 저장할 것인가 ?
```

```
저장 포맷: 이름,성적\n */
```

```
strncpy(buf, name, strlen(name));
```

```
//name 의 길이만큼 버프에다 name의 문자열을 복사한다
```

```
cur_len = strlen(buf);
```

//만약 name에 test가 입력되어 버프에 저장되었다면 strlen은 4이다. 하지만 배열은 0부터 시작하여 test의 마지막 t의 위치가 3인데 글자수를 읽었기 때문에 4가 나와 현재 test를 쓰고난 커서위치가 4가 참이되어 cur\_len 커서 위치가 4라는 정의를 내릴수 있다.

```
//printf("cur_len = %d\n", cur_len);
```

```
buf[cur_len] = ',';
```

//그 4의 위치에 ,를 추가한다. score 와 name을 구분하기 쉽기위해 넣어준 문자

```
sprintf(str_score, "%d", score);
```

//int 형 score 값을 char형 scor배열에 변환하여 저장한다.

```
strncpy(&buf[cur_len + 1], str_score, strlen(str_score));
```

//버프에 이름과 콤마가 저장되어있는데 그 다음 위치에 점수를 저장한다는 명령어이다.

여기서 점수를 char형으로 변환한 이유를 알 수 있다.

```
buf[strlen(buf)] = '\n';
```

//버프에 마지막 위치에 엔터를 추가하라는 명령어이다. 깔끔하게 보기위함이다.

```
//printf("buf = %s, buf_len = %lu\n", buf, strlen(buf));
```

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

//이제 버프에 저장된 값을 fd에 저장한다. fd는

```
//fd = open("score.txt", O_CREAT | O_EXCL | O_WRONLY, 0644)
```

//로 오픈 되어있기 때문에 결국 버프값은 score.txt에 저장된다. 저장되는 내용은

// 이름,점수\n이다.

```
close(fd);
```

//작업을 완료했기 때문에 충돌을 방지하기위해 fd를 닫아준다.

```
break;
```

```
case 3: //파일읽기
```

```
if((fd = open("score.txt", O_RDONLY)) > 0)
```

//파일읽기라는 메뉴는 저장된 파일이 있을 경우 실행할 것이다. 따라서 fd를 오픈할 때 리드온리로 읽기권한을 준다.

```
{
```

```
int i, backup = 0;
```

```
// 이름1,성적1\n
```

```
// 이름2,성적2\n
```

```
// .....
```

```
// 이름n,성적n\n
```

```
read(fd, buf, sizeof(buf));
```

//fd에 읽은값을 버프에 저장한다.

```
for(i = 0; buf[i]; i++)
```

```
{
```

```
if(!(strncmp(&buf[i], ",", 1)))
```

/\* 여기서 strncmp는 중요하다. 버프에 ,가 한 개 저장되었는게 참이면 0을 반환한다.

따라서 여기서 부정을 한다.if문이 참일 때 통과하기 때문 나머지는 수업시간에 선생님께 들었으므로 생략한다. \*/

```
{
```

```
strncpy(name, &buf[backup], i - backup);
```

```
backup = i + 1;
```

```
}
```

```

        if(!strcmp(&buf[i], "\n", 1))
        {
            strncpy(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;
}

```

## 수업내용 복습

### ---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); // 1번이 닫힘 따라서 모니터에 출력되는거 없음,
    dup(fd); // dup 복사하다 중복시키다. 따라서 종료된놈을 복사하게됨. 따라서 fd가 1번의 역할 a.txt
    에 기록됨
    printf("출력될까 ? \n");
    return 0;
}

```

fd를 a.txt를 생성하고 거기에 쓰기권한과 쓸때마다 초기화 옵션을 주었다.

출력을 달아버렸다.

dup(fd):를 하였다.

dup는 최근에 close 한걸 대체해준다.

따라서 fd가 출력을 대체해주었다.

a.txt 에 printf의 값이 저장된 것을 확인할 수 있었다.

### ---dup & gets---

```

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

int main(void)
{
    int fd;
    char buff[1024];
    fd = open("a.txt", O_RDONLY);
    close(0); // 입력을 닫았다. 키보드 입력받는걸 닫아서 gets를 썼었다.
    dup(fd);
    gets(buff); // 입력을 파일에서 받았다. 키보드에서 입력은 못받는다. 클로즈 한놈이 dup가 대체 >
    따라서 fd가 0을 대체해서 버퍼에 fd의 파일을 입력받은 버퍼를 출력하면 파일내용이 나오는 이유.
    // gets 는 클로즈 안했으면 화면에서 입력받게 해준다.
    printf("%s", buff);
    printf("출력될까?\n");
    return 0;
}
// dup를 두번하면 최근에 클로즈한걸 대체함 그래서 연속으로 두번하면 두개 다같은

```

gets 는 입력된 값을 받는다. 하지만 여기서 입력을 달아버렸다 그 달아버린 입력을 dup가 대체했다. 따라서 fd값이 buf에 저장된다.

fd를 바꾸면 buf도 바뀐다.

### ---dup & gets---

```
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ cat queue.c > ccc
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ cat < ccc
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

#define EMPTY 0

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

queue *get_node()
{
    queue *tmp;
    tmp=(queue *)malloc(sizeof(queue));
    tmp->link=EMPTY;
    return tmp;
}

void enqueue(queue **head, int data)
{
    if(*head==EMPTY)
    {
        *head = get_node();
        (*head)->data=data;
    }
    return;
    enqueue(&(*head)->link,data);
}
```

cat queue.c > ccc

cat < ccc 가 같은 루프라고 하는데 잘 이해가 안된다...

### ---512 lseek---

```
#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); //512번지를 a에 쓴다. master boot 레코드
    write(fd,&ch,1); //magic number 운영체제 마다 있는 어떤상황에도 512를 넘으면 안된다 옛날 운영체
    제는 그래서 매직넘버를 맨마지막에 할당한다 512를 넘지 않기 위해 이런식으로 했다. 보통 운영체제는 하위
    호환하는데 저코드를 버리지 않았다.
    close(fd);
    return 0;
}
```

512만큼의 크기가 있다고 할 때 배열은 0-511 이기 때문에 a를 512에 쓸려면 512-1을 해야한다. 이것을 하는이유는 좁은용량에서 마지막 부분을 찾기 위해서이다.

```
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ ./a.out mbr.txt
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ xxd mbr.txt
00000000: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000010: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000030: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000040: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000050: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000060: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000070: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000080: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000090: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000100: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000110: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000120: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000130: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000140: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000150: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000160: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000170: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000180: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000190: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000001a0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000001b0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000001c0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000001d0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000001e0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000001f0: 0000 0000 0000 0000 0000 0000 0000 0061 .....a
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$
```



### ---tail---

```
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ vi pipe.c
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ tail -c 20 queue.c
(head);
return 0;
}
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$
```

tail -c 20 xx.c 면 xx 파일에 끝에서 20글자를 보여준다.

```
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ tail -n 1 /var/log/messages
tail: cannot open '/var/log/messages' for reading: No such file or directory
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ tail -n 20 queue.c
printf("%d ",head->data);
head=head->link;
}
}

int main(void)
{
    queue *head = EMPTY;

    int i;
    int array[10]={6,5,8,12,1,23,2,7,27,24};

    for(i=0;i<10;i++)
    {
        enqueue(&head,array[i]);
    }
    printqueue(head);

    return 0;
}
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$
```

-n 20 은 xx파일의 20라인을 보여준다.

파일이 되게 많을 때 어떤 파일인지 쉽게 찾기 위한 방법이다.

### ---pipe---

```
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ ps -ef | grep bash
siyun    2261  2254  0 08:51 pts/6    00:00:00 bash
siyun    6075  2261  0 13:35 pts/6    00:00:00 grep --color=auto bash
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ ps -ef | grep bash | grep -v grep
siyun    2261  2254  0 08:51 pts/6    00:00:00 bash
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ vi pipe.c
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ ps -ef | grep bash | grep -v grep | awk '{print $2}'
2261
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$
```

ps -ef 하면 현재 동작중인 모든 프로세스가 나온다.

파이프 grep bash를 추가하면 터미널이랑 터미널을찾은 두 개의 프로세스가 나오  
고 grep -v grep 은 터미널을 찾은 프로세스를 제외하고 터미널 프로세스만 나  
온다. 여기서 awk print \$2 는 2번을 출력하는건데 2번이 pid 라 PID를 출력했  
다.

PID 란 사람에게 주민등록증이 있듯이 컴퓨터 프로세스의 고유 번호 식별 아이디  
이다.

```
siyun@siyun-Z20NH-AS51B5U: ~/my_proj/class20
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ ps -ef | grep bash | grep -v grep
siyun    2261  2254  0 08:51 pts/6    00:00:00 bash
siyun    6163  2254  0 13:41 pts/2    00:00:00 bash
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$
```

터미널을 한 개 더틀고 다시 grep를 해서 PID를 봤을 때, 서로 PID 가 다르다.  
따라서 사람들의 주민등록증이 다 다르듯 모든 프로세스의 PID는 다르다.

### ---fifo통신(myfifo block ver)---

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

```
int main(void)
```

```
{
```

```
    int fd,ret;
```

```
    char buf[1024];
```

```
    mkfifo("myfifo"); //myfifo라는 파일 생성
```

```
    fd = open("myfifo",O_RDWR); //fd를 myfifo를 읽는걸로 오픈
```

```
    for(;;)
```

```
{
```

```
    ret = read(0, buf, sizeof(buf));
```

//입력된걸 읽어서 버퍼에 저장 여기서 read는 block이기 때문에 입력이 있을때  
까지 기다린다.

```
    buf[ret-1] = 0; //이거는 잘 모르겠습니다.
```

```
    printf("Kryboard Input : [%s]\n",buf);
```

```
    read(fd, buf, sizeof(buf));
```

```
    buf[ret -1]=0;
```

```
    printf("Pipe Input : [%s]\n",buf);
```

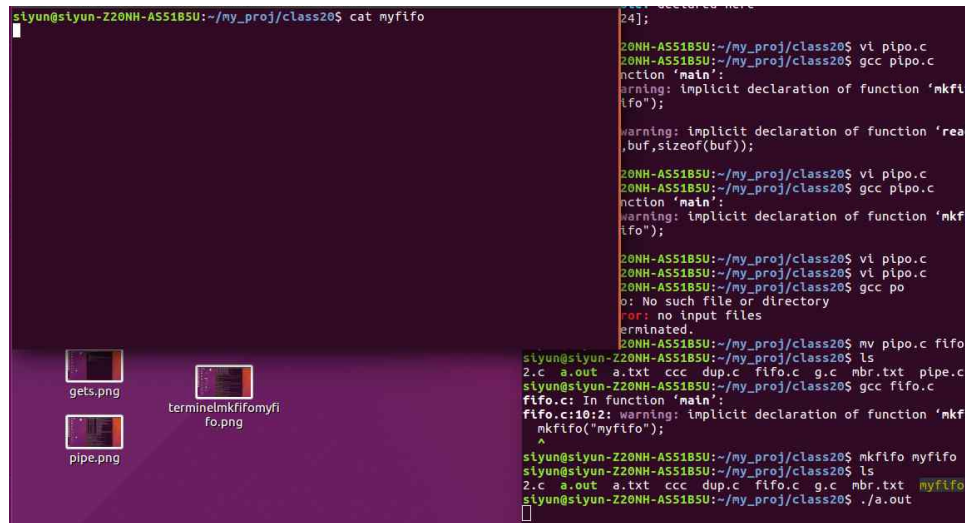
```

}
return 0;
}

siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ mkfifo myfifo
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ ls
2.c a.out a.txt ccc dup.c fifo.c g.c mbr.txt myfifo pipe.c queue.c quiz2.c tartest wr.c
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$

```

myfifo 라는파일을 만들어준다



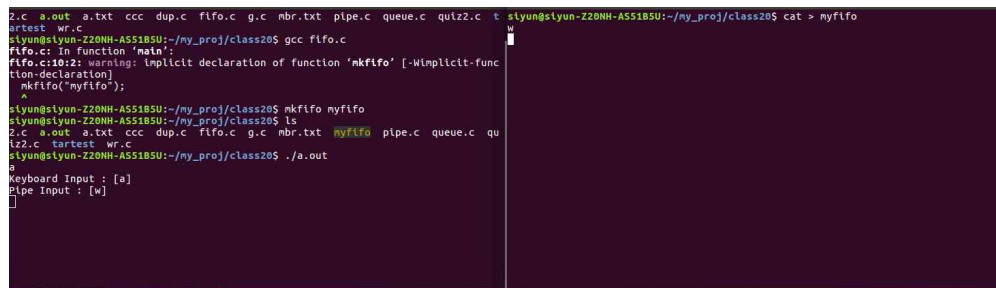
```

siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ cat myfifo
24j;
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ vi pipo.c
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ gcc pipo.c
pipo.c: In function 'main':
pipo.c:10:22: warning: implicit declaration of function 'mkfifo' [-Wimplicit-function-declaration]
pipo.c:10:22: warning: implicit declaration of function 'read' [-Wimplicit-function-declaration]
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ vi pipo.c
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ gcc pipo.c
pipo.c: In function 'main':
pipo.c:10:22: warning: implicit declaration of function 'mkfifo' [-Wimplicit-function-declaration]
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ mv pipo.c fifo
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ ls
2.c a.out a.txt ccc dup.c fifo.c g.c mbr.txt pipe.c
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ gcc fifo.c
fifo.c: In function 'main':
fifo.c:10:22: warning: implicit declaration of function 'mkfifo' [-Wimplicit-function-declaration]
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ mkfifo myfifo
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ ls
2.c a.out a.txt ccc dup.c fifo.c g.c mbr.txt myfifo
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ ./a.out
Keyboard Input : [a]
Pipe Input : [w]

```

./a.out을 하고 터미널 하나를 열어 cat > myfifo를 해준다.

여기서 cat > myfifo 라는게 cat 이라는 명령어는 myfifo를 읽고 화면에다 출력하는 기능인데 이게 통신이 가능한 건 open할 때 RDWR을 선언해서 인지 궁금하다.



```

2.c a.out a.txt ccc dup.c fifo.c g.c mbr.txt pipe.c queue.c quiz2.c tartest wr.c
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ cat > myfifo
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ gcc fifo.c
fifo.c: In function 'main':
fifo.c:10:22: warning: implicit declaration of function 'mkfifo' [-Wimplicit-function-declaration]
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ mkfifo myfifo
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ ls
2.c a.out a.txt ccc dup.c fifo.c g.c mbr.txt myfifo pipe.c queue.c
siyun@siyun-Z20NH-AS51B5U:~/my_proj/class20$ ./a.out
Keyboard Input : [a]
Pipe Input : [w]

```

read 가 블로킹이라서 한번밖에 안된다. 통신에 어울리지 않는다.

## ---fifo통신(myfifo nonblock ver)---

```

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

int main(void)
{
    int fd, ret;
    char buf[1024];
    fd = open("myfifo", O_RDWR);
    fcntl(0, F_SETFL, O_NONBLOCK);
    fcntl(fd, F_SETFL, O_NONBLOCK);
    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;
    }
}

```

//입력이 있는곳을 먼저 처리한다 읽을게 없으면 순회한다 논블록킹  
//블록킹 은 읽을게 있을때까지 기다린다.

여기서 0과 fd를 nonblock 옵션을 추가해주었다 그래서 read를 해주어도 block이 되지 않는다.

non블록은 기다리지 않고 돌다가 입력이 있을 경우 있는곳을 찾아가 먼저 실행해준다. 따라서 통신에 효율적이다

block fifo 와 똑같이 실행해주면 컴파일 결과를 확인할 수 있다.

block fifo는 컴파일을 출력한 터미널에서 먼저 입력해야했지만

nonblock은 어디서 입력하든 먼저 입력한 곳을 받아준다.

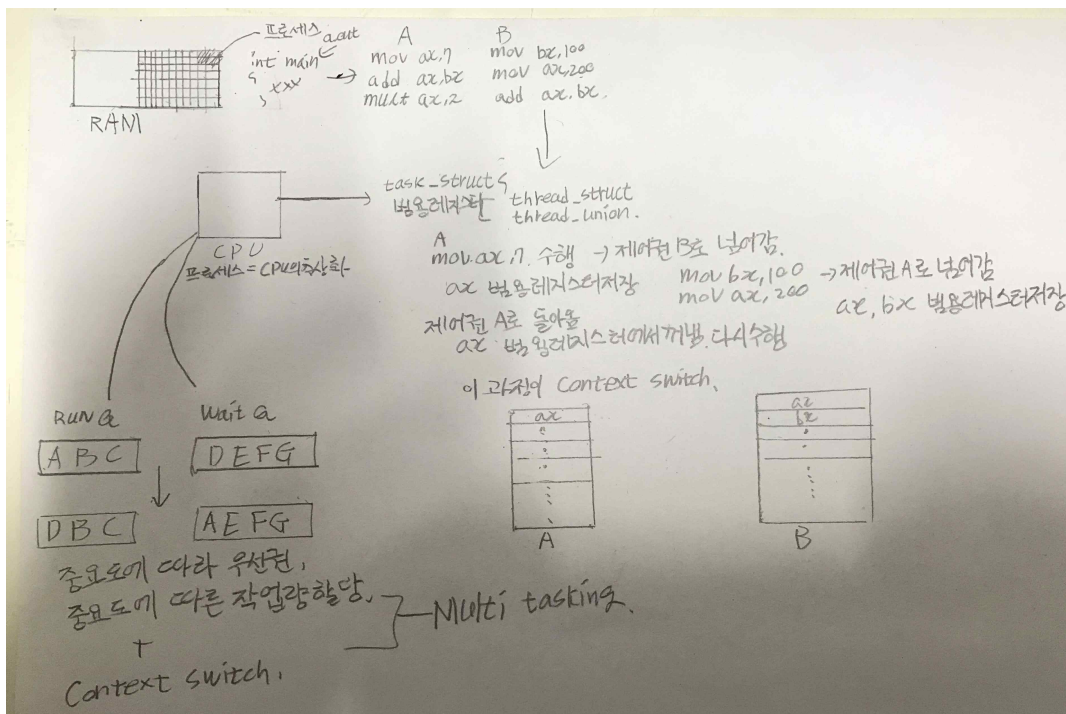


```

siyun@siyun-Z20NH-A551BSU:~/my_proj/class20$ gcc nonblock.c
/usr/lib/gcc/x86_64-linux-gnu/5/../../../../x86_64-linux-gnu/crt1.o: In function 'start':
(.text+0x20): undefined reference to 'main'
collect2: error: ld returned 1 exit status
siyun@siyun-Z20NH-A551BSU:~/my_proj/class20$ vi nonblock.c
siyun@siyun-Z20NH-A551BSU:~/my_proj/class20$ gcc nonblock.c
/usr/lib/gcc/x86_64-linux-gnu/5/../../../../x86_64-linux-gnu/crt1.o: In function 'start':
(.text+0x20): undefined reference to 'main'
collect2: error: ld returned 1 exit status
siyun@siyun-Z20NH-A551BSU:~/my_proj/class20$ vi nonblock.c
siyun@siyun-Z20NH-A551BSU:~/my_proj/class20$ gcc nonblock.c
siyun@siyun-Z20NH-A551BSU:~/my_proj/class20$ ./a.out
Keyboard Input : [1]
Pipe Input : [2]
Pipe Input : [3]
Pipe Input : [e]
Pipe Input : [e]
Pipe Input : [s]
Pipe Input : [f]
Pipe Input : [dse]

```

## 과제 - Context-Switchin , Multi tasking 그림 설명



CPU

CPU

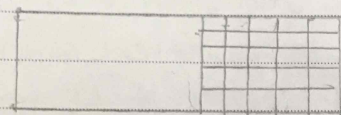
프로세스 (CPU의 추상화)

프로세스는 main 같은걸로 우리가 만든 프로그램의 집합.

CPU 주파수가 2GHz 일때,

$$f = \frac{1}{T} \therefore T = \frac{1}{f} \Rightarrow \frac{1}{2\text{GHz}} = \frac{1}{2 \times 10^9} = 5 \times 10^{-10}$$

우리가 부르는 메모리는 RAM.



RAM. ↑ 데이터가 저장되는 부분

프로그램 컴파일 했을때

```

a.out {
    int main
    {
        while(1)
        {
            push
            mov
            add
            call
            ret
        }
    }
}

```

→ 레지스터를 제어.

\* 여기서 CPU는 1 clock에 한가지 연산밖에 하지 못하는데 어떻게 여러 프로세스가 동시에 동작할까?

→ Context switching, Multi-Tasking

\* CPU의 clock 주파수가 2GHz 일때

주기는  $5 \times 10^{-10}$ 이다. 이거 아주 빠른 속도로 제어권을

넘겨주면서 CPU를 사용한다만 우리가 느끼지 못하는 시간에 모든 작업이 완료된다.

DATE . .

task struct 레지스터  
포함한 곳에 저장할 때  
여기서 여기서 우선순위를

A	B
mov ax, 7	mov bx, 100
add ax, bx	mov ax, 200
mult ax, 2	add ax, bx

만약 두개의 프로세스 A, B가 위와 같은 작업을 한다고 가정

A의 mov ax, 7 을 수행하고 B로 제어권이  
넘어가 B가 수행되고 A로와서 A를 수행하면  
결과가 이상하다. 그래서 task\_struct 안에  
레지스터를 제어하는 thread\_struct, thread\_union  
에 A의 중간 수행값과 B의 중간 수행값을 남겨주  
고 다시 A 또는 B에 제어권이 넘어올 때 중간까지  
값을 레지스터에서 꺼내서 작업을 수행.

이게 Context-switching

여기서 우선순위를 정해주는 run queue 와  
wait queue 에 작업량을 중요도에 따라  
균등하게 나눠주는거까지 포함하면  
Multi-Tasking.

## 용어정리

**man -s2 open** 시스템 콜에 대한 명령어를 볼수 있음

**O\_APPEND** 계속해서 이어쓴다. (성적 관리 프로그램에서 SEEK\_CUR을  
SEEK\_END로 보내버림)

**strncpy(저장할곳, 복사할내용, 복사할크기);** string을 복사한다.

**strlen(char 변수);** string 의 길이를 측정한다

**strncmp(a[i], "!", 1);** string을 비교한다. (a라는 배열 I번째에 !라는 글자가 1개 있  
으면 0을 리턴한다)

**man** 궁금할 놈을 알려주는 착한놈이다.