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

날 짜 : 2018 . 3. 21

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

Quiz. 2

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. 입력한 이름과 성적은 파일에 저장 되어야 한다.
5. 당연히 통계 관리도 되어야 한다(평균, 표준 편차)
6. 프로그램을 종료 하고 다시 키면 이제 파일에 서 앞서 만든ㄷ 정보들을 읽어와서내용을 출겨해줘야 한다.
7. 언제든 원하면 내용을 출력할 수 있는 출력함수를 만든다.
  (특정 번튼을 입력하면 출력이 되게 만듬)
(역시 System Call 기반으로 구현하도록 함)

```
// 1. 학생 성적을 입력 하는 함수이다.
void disp_student_manager(int *score, char *name, int size)
{
        char *str1 = "학생 이름을 입력하시오: ";
        char *str2 = "학생 성적을 입력하시오: ";
        char tmp[32] = {0};

        // 위에 입력한 str1 을  '기본출력'  해준다.
        write(1, str1, strlen(str1));
        // 가지고온 name 의 주소를 입력 값으로 받는다.
        read(0, name, size);
        // 한번더 반복한다.
        write(1, str2, strlen(str2));
        read(0, tmp, sizeof(tmp));

        // ascii to integer 를 통해서 score 에 문자를 숫자로 받아낸다.
        *score = atoi(tmp);
}
```

```
// 2. 확인용으로 터미널 창에 출력해 준다.
void confirm_info(char *name, int score)
{
	printf("학생 이름 = %s\n", name);
	printf("학생 성적 = %d\n", score);
}
// 3. 중간에 개행이 나오면서 문자열이 이상해지는것을 막기 위해 \n 일 때 \0 으로 바뀌게 한다.
void remove_enter(char *name)
{
	int i;

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


//3. main 에서의 동작.
int main(void)
{
	// Slab 할당자가 32 byte 를 관리하기 때문에 성능이 빠름
	char name[32] = {0};
	char str_score[32] = {0};
	char buf[64] = {0};
//for 문을 통해서 계속 동작을 할 수 있게 해준다. 스위치로 계속 동작을 확인을 위해.
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:
				// 위에 만든 함수를 그대로 보여준다.
				disp_student_manager(&score, name, sizeof(name));
				// 중간에 개행이 보기 안좋아 넣어줌.
				remove_enter(name);
				// 확인용으로 프린트를 한번 해준다.
				confirm_info(name, score);
				// 스택에 입력한 것을 저장한다.
				enqueue(&head, name, score);
				// 스택에 저장된것을 출력한다.
				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)
                        // append 를 이용해서 이어붙이기를 한다.
                        fd = open("score.txt", O_RDWR | O_APPEND);

            /* 어떤 형식으로 이름과 성적을 저장할 것인가 ?
              저장 포맷: 이름,성적\n */
            // strncpy (복사할곳, 복사할 거, 몇개) (buf<-name)
            strncpy(buf, name, strlen(name));
            // 현재 위치를  저장한다. buf 의 길이가 결국 현재 위치
            cur_len = strlen(buf);
            //printf("cur_len = %d\n", cur_len);
            // , 로 구분을 위에 넣는다.
            buf[cur_len] = ',';
            //  숫자를 저장하기 위해서 sprint 로 문자로 바꿔준다.
            sprintf(str_score, "%d", score);
            //  이제 앞에 저장된 위치에 뒷부분 부터 저장을 위해 +1 부터 score 를 받는다.
            strncpy(&buf[cur_len + 1], str_score, strlen(str_score));
            // 마지막으로 개행이 되게 한다.
            buf[strlen(buf)] = '\n';
            //printf("buf = %s, buf_len = %lu\n", buf, strlen(buf));

            // 저장한것을 열어 놓은 score.txt 안에 쓴다.
            write(fd, buf, strlen(buf));

            close(fd);

            break;

case 3:
            // 읽기 위해 파일을 연다.
            if((fd = open("score.txt", O_RDONLY)) > 0)
            {
                        int i, backup = 0;
                        // 이름 1,성적 1\n
                        // 이름 2,성적 2\n
                        // .....
                        // 이름 n,성적 n\n
                        // 파일의 내용을 읽는다.
                        read(fd, buf, sizeof(buf));

                        for(i = 0; buf[i]; i++)
                        {
                        // 안에 있는 내용을 다 돌려서 본다.
```

```
                                        // 비교함수로 ‘,’가 오면 이름을 출력하고 백업 위치를 저장
                                                if(!(strncmp(&buf[i], ",", 1)))
                                                {
                                                        strncpy(name, &buf[backup], i - backup);
                                                        backup = i + 1;
                                                }
                                        // 비교함수로 ‘,’가 오면 성적 값만 뽑아 출력하고 백업위치를 저장.
                                                if(!(strncmp(&buf[i], "\n", 1)))
                                                {
                                                        strncpy(str_score, &buf[backup], i - backup);
                                                        backup = i + 1;
                                                        enqueue(&head, name, atoi(str_score));
                                                }
                                        }
                                        // 저장한 것을 출력 해서 보여준다.
                                        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;
}
```

* man -s2 함수명 =시스템 콜 관련 명령어, 함수 등의 메뉴얼을 알려준다.
* man 함수명 = 명령어 , 함수 등의 메뉴얼을 알려준다.

lseek(fd , 0 ,SEEK_CUR) → 현재 일기/ 쓰기 포인터 위치.
SEEK_SET → 파일의 시작.
SEEK_END → 파일의 끝.

위의 0 위치가 뒤에 seek 가 가리키는 부분에서 + 혹은 - 몇에서 시작할지 정해준다.
→ 맨뒤 옵션에 따라서 offset 위치에서 시작한다.
→ lseek 의 반환값을 이용해 크기를 구할 수 있다. lseek(fd , 0 ,SEEK_END) 라 하면 처음부터 끝이기에 크기라고 보면 된다.

→ lseek 가 좋은 점은 위치 상태를 저장하기 때문에 반복 입력 삽입시에 유용하게 사용할 수 있다.

1. dup() →

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

int main(void)
{
	int fd;
	//fd 는 3 이다. a.txt 를 열어 둔다.
	fd = open("a.txt",O_WRONLY|O_CREAT|O_TRUNC,0644);
	// 기본 출력을 닫는다.
	close(1);
	// dup(fd) fd 쪽으로 복사하기.
	dup(fd);
	// 그럼면 기본 출력이 되는 곳이 막혀 있기 때문에 fd 가 3 번이니 그쪽으로 저장된다.
	printf("출력될까 ? \n");

	return 0;
}
```

2. dup() →

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

int main(void)
{
        int fd;
        char buff[1024];
        //fd 는 3 이다. a.txt 를 열어 둔다.
        fd = open("a.txt",O_RDONLY);
        // 기본 입력을 닫는다.
        close(0);
        // dup(fd) fd 쪽으로 복사하기.
        dup(fd);
        // gets 가 화면에서 입력을 받게 해준다.
        gets(buff);
        // 기본 입력이 닫혀있기 때문에 3 번으로 복사 된다.
        printf("%s\n",buff);
        printf("출력될까?\n");
                return 0;
}
```

< 리다이액션 >

cat 파일명 > 복사할 파일명 하면 왼쪽에서 오른쪽으로 내용 복사.

cat < ccc 하면 ccc 에 있는 것을 화면에 출력하겠다는 뜻이다.

< gets() >

화면에 입력 받게 해준다. 엔터를 하면 입력이 실행.

Pipe communication

*** ps 명령어는 process 의 state 를 보는 명령어에 해당한다.**

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

-print $2 : 내고유 식별 번호를 확인 가능하다. PID

< tail >

명령어는 파일 내용의 마지막 부터 읽을 때 주로 사용한다.

*tail -c 숫자 : 글자를 숫자만큼 뒤에서 부터 뿌림.

*tail -n 숫자 : 한줄을 숫자만큼 뒤에서 부터 뿌림.

@안에 저장된 마지막 위치를 변경했었다. (매직 넘벌)

./a.out mbr.txt → xxd mbr.txt (master boot record)

* 플로피 디스크(512byte) : magic nuber → 마지막 510 쯤에 들어있는 번호. 운영체제 번호가 들음

< fifo >

(pipe()에서 생성한 파이프를 이용하는 것은 부모와 자식 프로세스에서만 사용됩니다. 그러나 FIFO 를 이용하면 서로 다른 프로세스에서 사용할 수 있으며, FIFO 를 생성하는 파일 이름을 알고 있다면 누구나 사용할 수 있음.)

1. mkfifo myfifo
2. 실행 파일 실행.
3. 다른창 cat > myfifo

ls -al /dev 맨앞이 'd' 디렉토리
'p' 파이프
'c' 케릭터 - 순서가 있음, 키보드, 모니터 등의 순서가 필요하게 돌아가는 것들.
's' 소켓
'b' 블록킹 – 특정단위, 순서가 필요 x(d-ram)
(메모리의 최소 단위 (페이지) = *4kbyte)

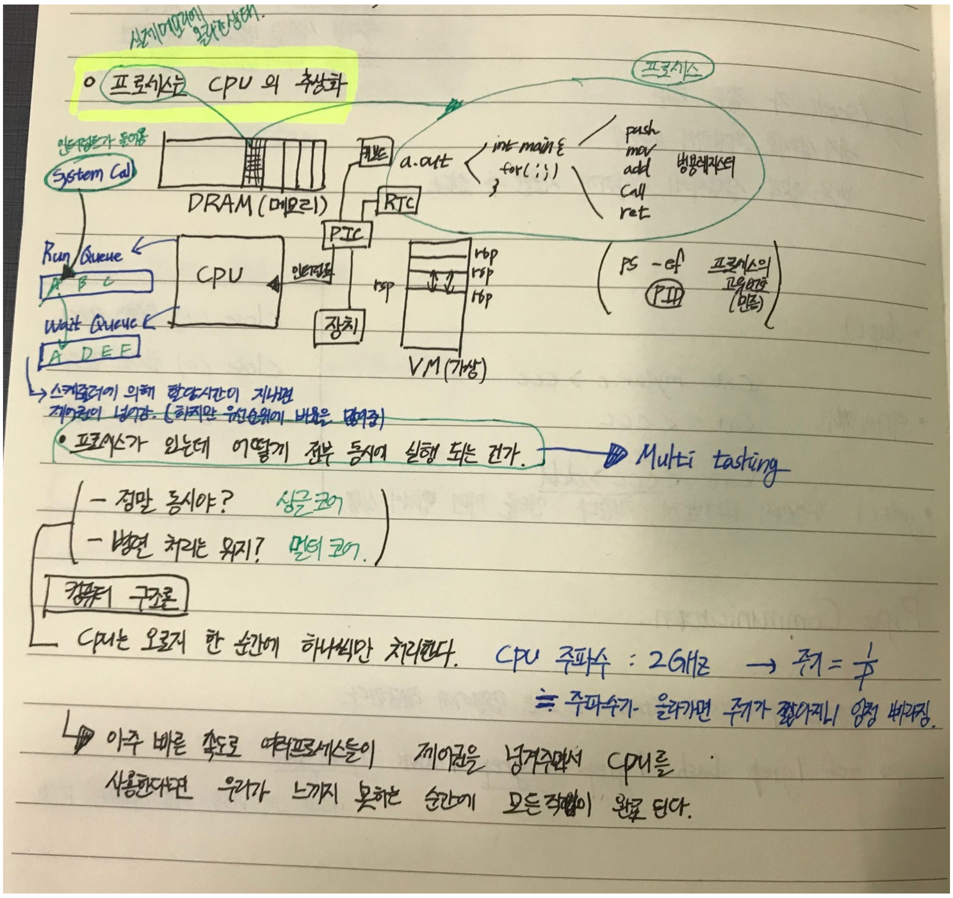
< fcntl(0, F_SETFL, O_NONBLOCK) >

*F_SETFL - 읽을게 없을 때 권한을 넘긴다.

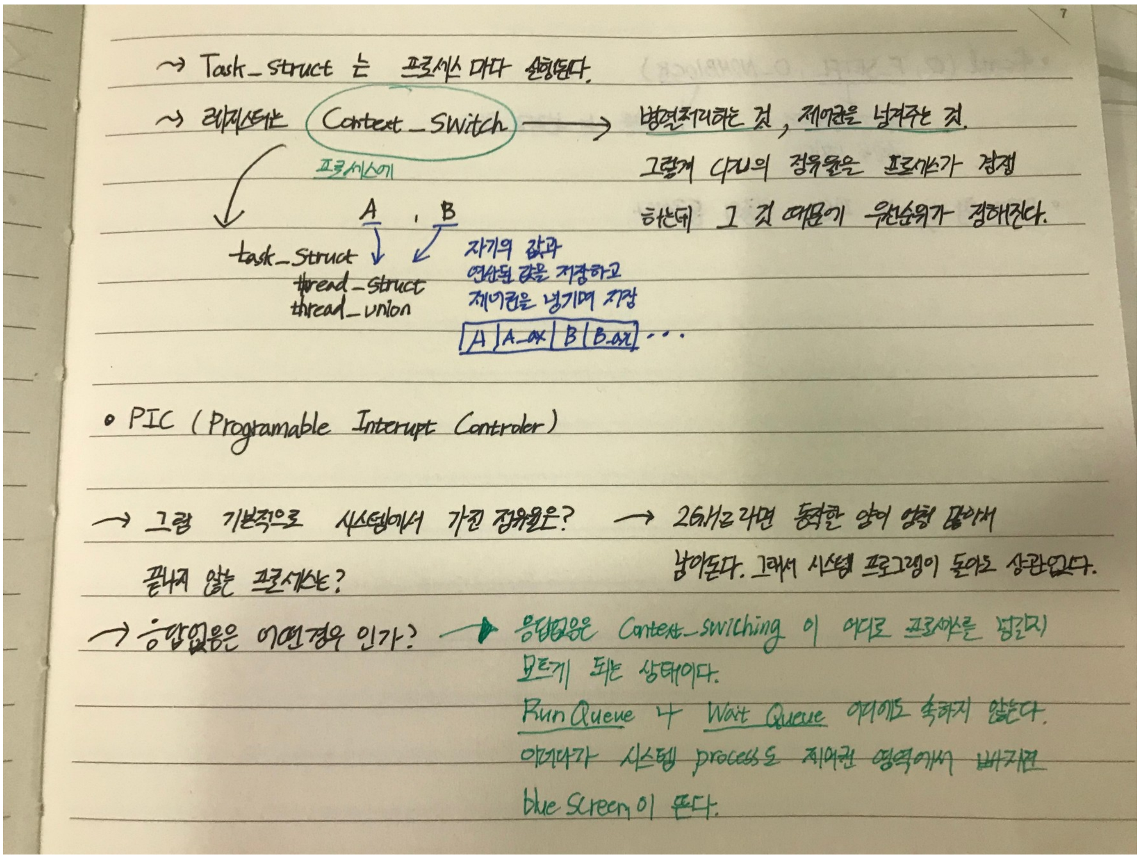
* O_NONBLOCK - 디스크립터 0 번을 블록킹 하지 않는다는 뜻이다.

< PIPE 는 파일의 일종의 통로이다. >

<프로세스의 추상화>



실제메모리에 올라간상태.
○ 프로세스는 CPU 의 추상화
프로세스
인터럽트가 들어옴
System Call
DRAM(메모리)
키보드
a.out
int main {
for(;;)
}
push
mov
add
call
ret
범용레지스터
RTC
PIC
Run Queue
A B C
CPU
인터럽트
장치
rsp
rbp
rsp
rbp
Wait Queue
A DEF
VM(가상)
ps -ef
PID
프로세스의 고유번호 (민증)
스케줄러에 의해 할당시간이 지나면 제어권이 넘어감. (하지만 우선순위에 따라 달라짐)
• 프로세스가 있는데 어떻게 전부 동시에 실행 되는 건가.
Multi tasking
- 정말 동시야? 싱글코어
- 병렬 처리는 뭐지? 멀티코어.
컴퓨터 구조론
CPU는 오로지 한 순간에 하나씩만 처리한다.
CPU 주파수 : 2GHz → 주기 = 1/f
∴ 주파수가 올라가면 주기가 짧아지니 엄청 빠라짐.
아주 빠른 속도로 여러프로세스들이 제어권을 넘겨주면서 CPU를 사용한다면 우리가 느끼지 못하는 순간에 모든 작업이 완료 된다.



7
→ Task_struct 는 프로세스 마다 실행된다.
→ 레지스터는 Context_Switch → 병렬처리하는 것, 제어권을 넘겨주는 것.
프로세스에
그렇게 CPU의 점유율은 프로세스가 경쟁하는데 그 것 때문에 우선순위가 정해진다.
A , B
task_Struct
thread_struct
thread_union
자기의 값과 연산된 값을 저장하고 제어권을 넘기며 저장
A | A_ax | B | B_ax ...
• PIC (Programable Interupt Controler)
→ 그럼 기본적으로 시스템에서 가진 점유율은? 끝나지 않는 프로세스는?
→ 2GHz 라면 동작한 양이 엄청 많아서 남아돈다. 그래서 시스템 프로그램이 돌아도 상관없다.
→ 응답없음은 어떤경우 인가?
응답없음은 Context_switching 이 어디로 프로세스를 넘길지 모르게 되는 상태이다.
Run Queue 나 Wait Queue 어디에도 속하지 않는다.
여기다가 시스템 process도 제어권 영역에서 빠지면 blue Screen이 뜬다.