**Pintos Project 1 : User Program Basic**

**(설계 프로젝트 수행 결과)**

**:**

과목 명 :운영체제

담당 교수 :김영재

조 / 조원 : 10조/황순, 허준형

개발 기간 :2018.10.01~2018.10.16

**프로젝트 제목 : pintos project 1 user program basic**

**제출일 :2018.10.16**

**참여 조원 : 황순, 허준형**

1. **개발 목표**

* Pintos에는 기본적인 User Program 실행환경은 구성되어 있다. 그러나 Argument Passing, 과 system call 등의 기능이 갖춰져 있지 않다. 이번 프로젝트에서 우리는 이 두가지 기능을 구현하고, process간 synchrnoization을 구현하여 프로세스간 생성을 구현한다.(busy waiting 기법)

1. **개발 범위 및 내용**
   1. **개발 범위**

* 1. Argument Passing 및 Parsing
* 주어진 argument를 공백문자를 기준으로 parsing하여 stack에 적절히 저장하도록 한다.
* 2. User Memory Access
* user program 에서 적합하지 않은 메모리 영역을 접근하는 경우, 적절히 경우를 나누어 종료시켜야 한다. 주로 다음과 같은 세가지 경우가 있다.(매뉴얼 참조)
* i). 물리적 주소와 총 스택사이즈 (64MB) 를 벗어나는 경우.
* Ii). 적절한 page를 가져왔는지 검사하는 것.
* Iii). 잘못된 포인터( 예를 들어 NULL Pointer)를 참조할 경우.
* 3. System Call
* -어떤 경우에 적절한 system call을 구축할지 결정한다.
* 다음과 같은 명령어들을 차례로 구축한다.
* -halt(): devices.c 에 정의된 shutdown\_power\_off()를 호출하여 종료시킨다.
* -exit(): process의 종료상태를 출력하고, 적절히 process의 상태를 종료시킨다.
* -wait: process들을 생성할경우 ( 부모, 자식) 그 둘이 적절히 종료되거나 수행될때 프로세스를 기다리게 한다.
* -write: 출력 내용을 stdout으로 출력한다.
* -read: 파일로부터 읽은 내용을 버퍼에 복사한다.
  1. **개발 내용**
* **1. Page 의 개념**
* **-**A page, sometimes called a virtual page, is a continuous region of virtual memory 4,096 bytes (the page size) in length. (pintos manual pp.39)
* 가상메모리의 4096바이트의 길이를 가진 주소이다.
* **2. Thread의 개념**
* **-**A basic unit of CPU utilization (or scheduling) or a flow of control within a process, and consists of a thread ID, a PC, a register set and a stack space (similar to process).

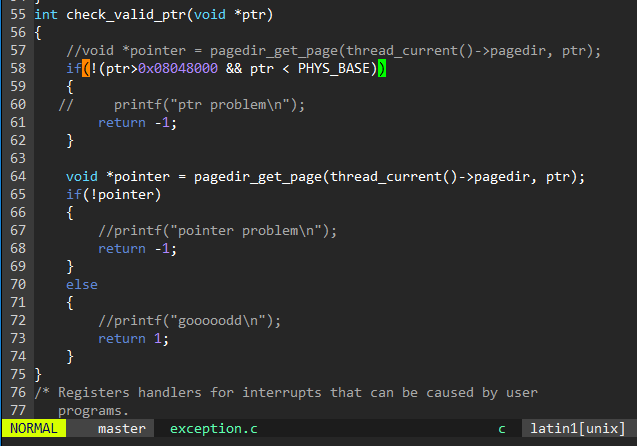
프로그램의 플로우를 저장하는 것으로, 핀토스에서는 doubly linked list로 구현되어있다.

* **3. User 에서 kernel 메모리의 접근을 막는 방법**
* You need this ability before you can even obtain the system call number, because the system call number is on the user’s stack in the user’s virtual address space. This can be a bit tricky: what if the user provides an invalid pointer, a pointer into kernel memory, or a block partially in one of those regions? You should handle these cases by terminating the user process.

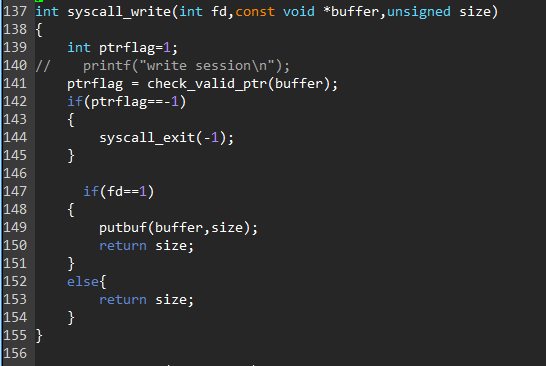
(pintos manual pp,32) 이다. 해석하자면 system call number가 user의 가상메모리 주소로 쌓이는데, 이것이 실수로 커널메모리를 가리키면 큰 문제가 된다. 핀토스 매뉴얼에서는 다음과 같은 방식을 제안한다.

첫번째는 user\_provided pointer가 적절한지 체크한후, 그 포인터를 해제하는 것이다. (pintos manual pp.27)

두번째는 user pointer가 PHYS\_BASE 아래인지 확인하여 체크하는 것이다.(핀토스 매뉴얼 pp.27) 우리는 이 방법을 선택하였다.  
→User program 은 많은 종류의 argument를 가진다. 구체적인 코드 체크 방식은 다음과 같다.

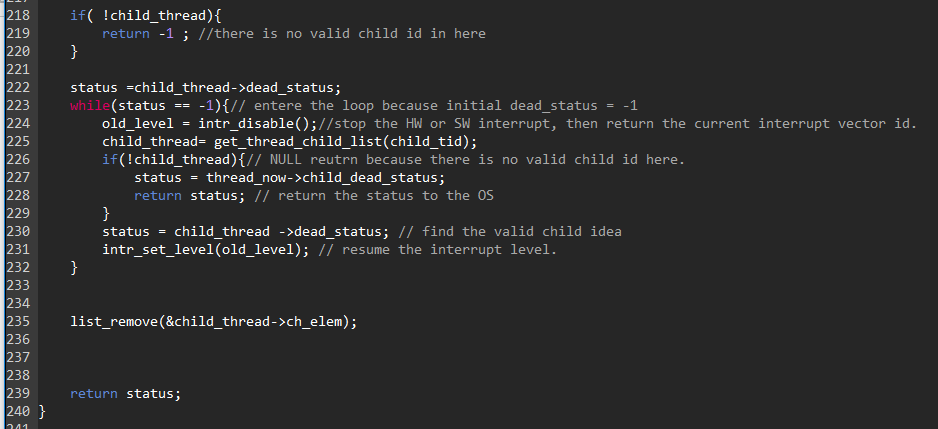
****

이를 통해 시스템 콜을 체크한 모습은 다음과 같다.

****

**3. 동기화 기법.**

본 작성자들은 동기화 기법으로 busy waiting을 사용하였다. Process\_wait 에서 자식 프로세스가 생성되면, 프로세스가 끝날때까지 기다리는 방법으로, 시간이 조금 느리다는 단점이 있지만 가장 직관적으로 구현할 수 있다. 코드는 다음과 같다.

****

**User progr 실행순서**

* -> 명령어 argument를 공백을 기준으로 구분한다. -> 이 argument를 80x86 의 메모리 규칙에 맞추어 스택(ESP)에 적절히 쌓는다.
* userprogram에서 넘긴 포인터(주소)를 체크하여 kernel영역을 참조하거나, 혹은 PHYS\_BASE를 초과하는 영역을 접근하는지 확인하여 종료한다.
* 마찬가지로 return address 보다 위의 메모리를 접근하는지 확인한다.
* User memory가 0x0804800 보다 아래를 참조하고 있는지 확인한다. 이는 64MB로 핀토스의 최대 사이즈이다.
* NULL pointer를 접근하는지 확인하여 종료시킨다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

|  |  |  |
| --- | --- | --- |
| * **구분** | * **기간** | * **비고** |
| * **Argument** * **Passing** | * **2018.10.01-** * **2018..10.02** | * **Manual을 정독하고, 프로그램의 순서도와 흐름을 파악하는데 중점을 둠** * **MakeESP함수와**   **parsingFilename함수를 구현하였다.** |
| * **‘’SYSCALL** | * **2018. 10. 04 ~** * **2018.10.05** | * **구현방식의 논의 및 회의.**   **halt, exit, 를 구현하였다.**  **Wait 구현시 busy waiting방식을 매뉴얼을 통해 숙지하였다.** |
| * **UserMemory Access** | * **2018.10.07 ~ 2018. 10.09** | * **- 매뉴얼에 나와있는 get\_user를 통해 적절한 주소를 받아오기로 하였다.** * **이를 통해 메모리 참조의 잘못된 경우를 코딩하였다.** |
| * **Debugging** | * **2018.10.10 ~ 제출일전까지** | * **Signed 와 unsigned의 type 전환 문제, little edian과 big edian에서 발생한 문제를 해결하였다.** |

* 1. **개발 방법**

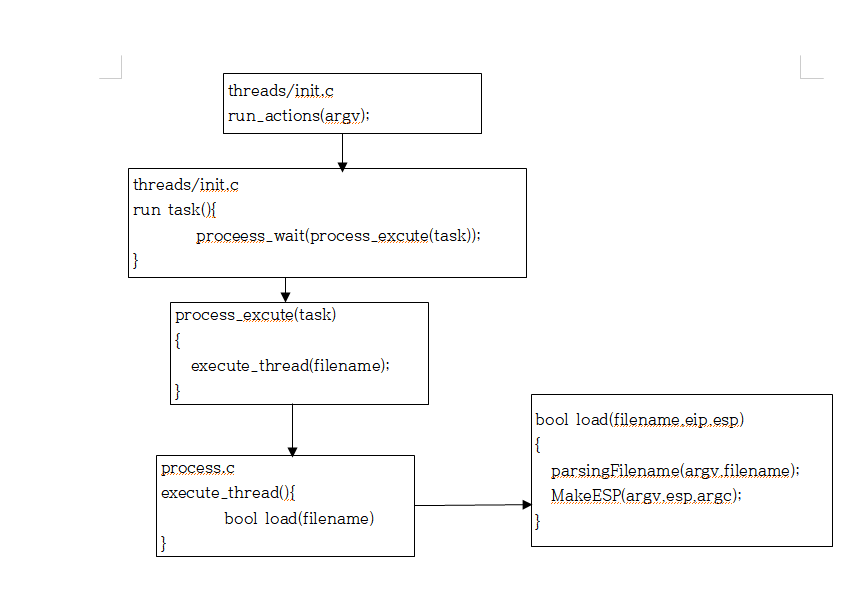
**전통적인 agile 방식이 아닌 github를 이용한 협업을 하였다. 모든 코드를 거의 함께 구현하면서 개발하였기 때문에 서로가 서로의 코드를 완벽히 이해할 수 있었다. 특히 디버깅시 어떤 메시지를 할것인지도 완벽히 알고 있어서 정확히 두배의 효율을 낼 수 있었다.**

* 1. **연구원 역할 분담**

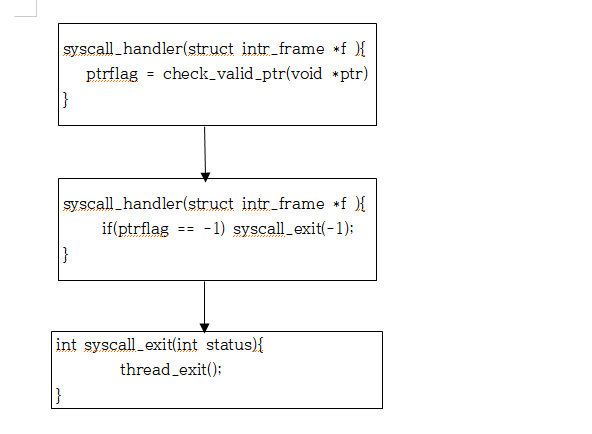
|  |  |
| --- | --- |
| * **조원** | * **내용** |
| * **황순** | * **Argument passing 구현, systemcall 함수 구현, 추가 구현( sum, Fibonacci) 완성** |
| * **허준형** | * **Argument passing 구현,system call 함수 구현, busy\_waiting 완성** |

1. **연구 결과**
   1. **합성 내용**

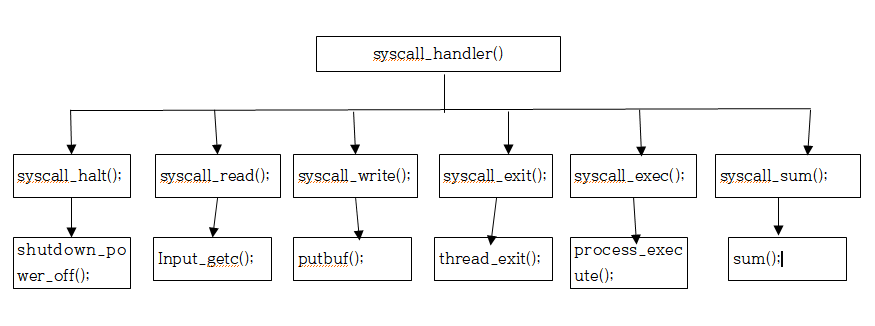
**- Argument Passing**

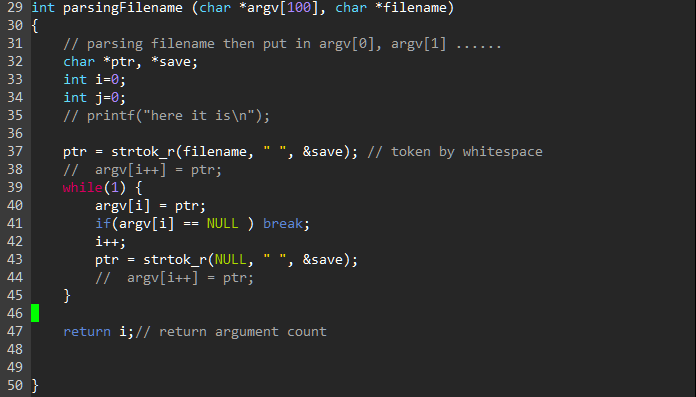


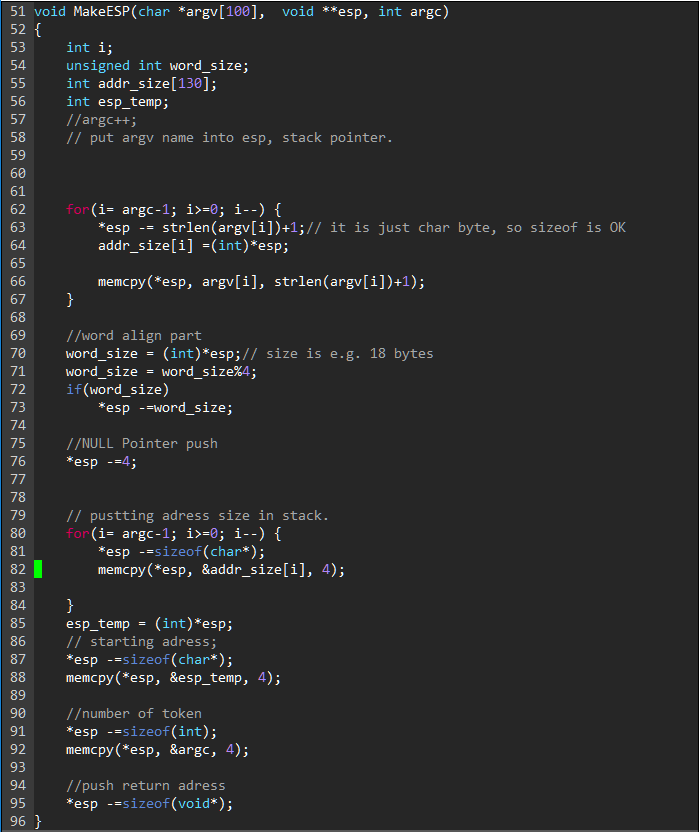
**- User Memory Access**



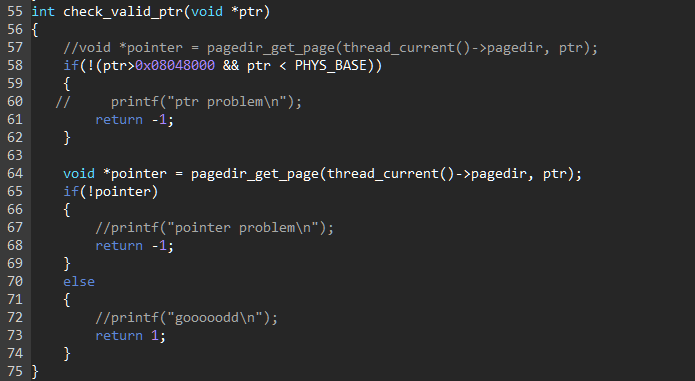
**- System Call Infrastructure & Implementation**

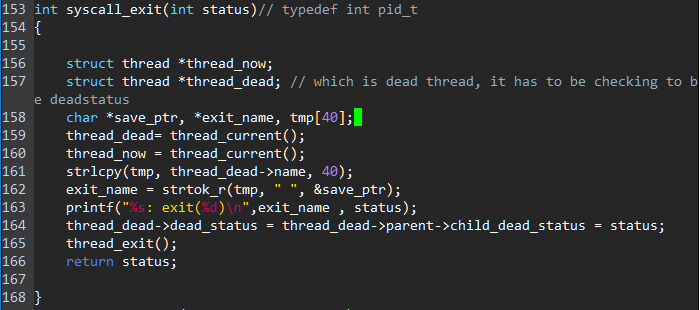
* 
  1. **제작 내용**
* **userprog/process.c**





* **User Memory Access**





**-System Call Infrastructure & Implementation**

(1) syscall\_halt () : userprog / syscall.c

syscallnum이 SYS\_HALT인 경우 syscall\_halt()함수 호출

(2) syscall\_read() : userprog / syscall.c

syscallnum이 SYS\_READ인 경우 user\_read()함수 호출

fd가 0일 때 input\_getc()함수 수행 후 count를 return한다.

(3) syscall\_write() : userprog / syscall.c

syscallnum이 SYS\_WRITE인 경우 user\_write()함수 호출

fd가 1일 때 putbuf()함수 수행 후 size를 return한다.

(4) syscall\_exec () : userprog / syscall.c

syscallnum이 SYS\_EXEC인 경우 syscall\_exec()함수 호출

(5) syscall\_wait() : userprog / syscall.c

syscallnum이 SYS\_WAIT인 경우 syscall\_wait()함수 호출

(6) syscall\_exit() : userprog / syscall.c

syscallnum이 SYS\_EXIT인 경우 syscall\_exit()함수 호출

(7) syscall\_sum() : userprog / syscall.c

syscallnum이 SYS\_SUM\_4인 경우 sum()함수 호출

**thread.h**

struct thread\* parent;

int dead\_status;

bool dead;

bool exist\_childlist;

int child\_dead\_status;

struct list child\_list;

- parent : 현재 thread의 부모를 가리킨다

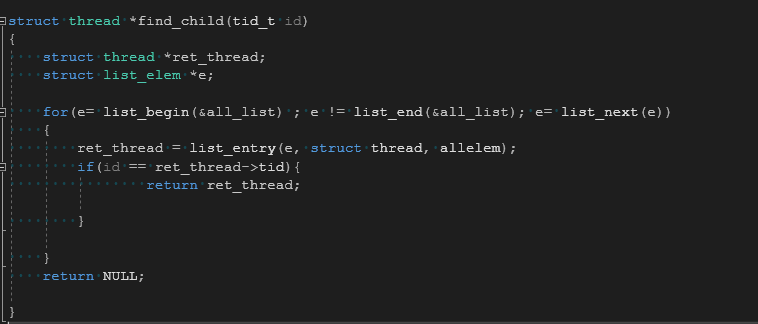
- dead\_stauts : 현재 쓰레드가 죽었는지 살았는지를 표현해주는 flag이다.

- dead : 현재 쓰레드가 죽었는지 살았는지를 표현해주는 flag이다.

- bool exist\_childlist : child가 있는지 표현해주는 flag이다.

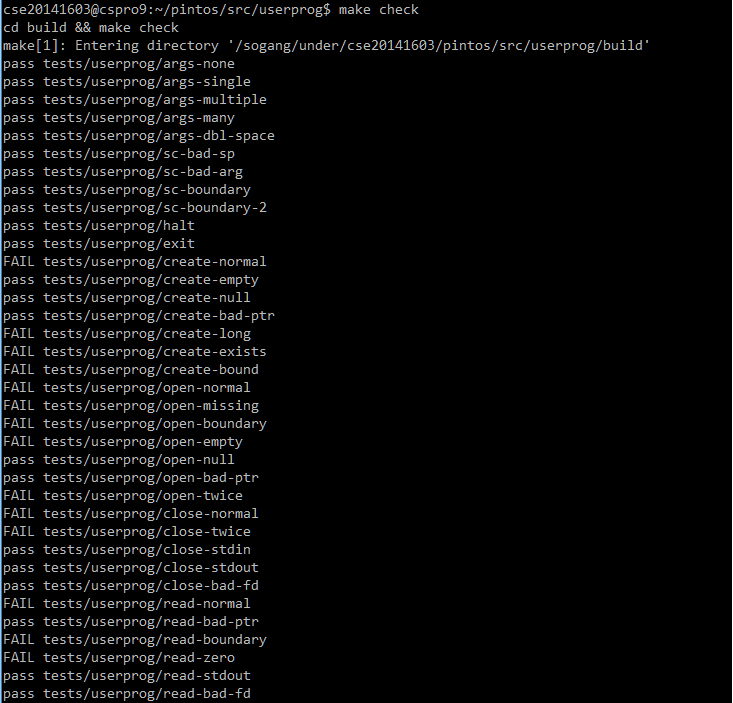
- int child\_dead\_status; : child가 죽었는지 표현해주는 flag이다.

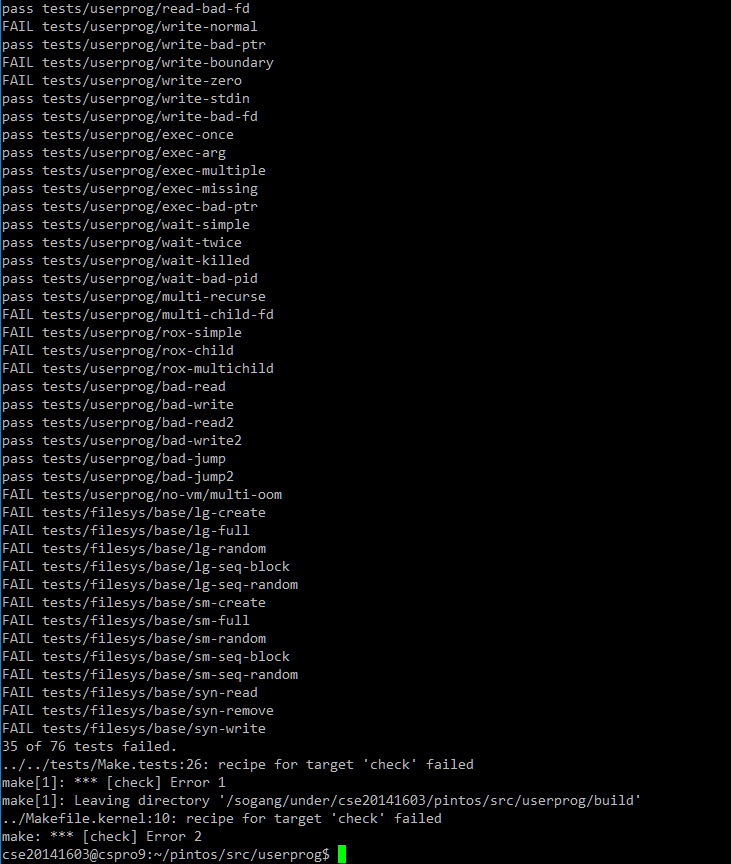
- struct list child\_list : child의 리스트를 보여주는 child list 이다.



**현재 쓰레드에 존재하는 child를 찾아주는 함수이다.**

**C. 시험 및 평가 내용**



* + 

1. **기타**
   1. **연구 조원 기여도**

* **허준형(50%) , 황순(50 %)**
  1. **소감**

**허준형- 이번 핀토스 프로젝트를 통해서 OS 가 어떤 식으로 명령어를 받아서 프로세스를 형성하는지 알 수 있었다. 특히 이번기회로 여러 동기화 기법에 대해 깊이 공부 할 수 있었고, 거대한 프로그램은 개발을 위해 어떤식으로 구분하고 쓰는지 알 수 있었다. 무엇보다 팀원과의 의사소통이 중요함을 깨달을 수 있었고, 다른 사람과 함께 연구하고 공부하며 하는 것이 굉장히 효율이 높은 법이라는 것도 알 수 있었다. 다음 두번째 프로젝트도 이렇게 열심히 하려고한다.**

**황 순 : 처음에 pintos manual과 주어진 code를 보고 분석하는데 오랜 시간이 걸렸다. 작성한 코드가 지금까지의 과목과는 출력 결과를 한번에 확인하기가 힘들어서 디버깅 할 때 적응하느라 애를 먹었다. 그렇지만 매뉴얼을 참고하면서 코드를 작성하다보니 어느정도 운영체제에 대하여 이해하는 계기가 되었던 것 같다.**