

Pintos 프로젝트 2_1. User Program Basic

(설계 프로젝트 수행 결과)

과목명 : [CSE4070-01] 운영체제
담당교수 : 서강대학교 컴퓨터공학과 박성용

조원 : 83조 권명준, 전해성
개발기간 : 2017. 09. 30. -2017. 10. 20.

최종보고서

프로젝트 제목: Pintos 프로젝트 2_1. User Program Basic

제출일: 2017. 10. 20.

참여조원: 83조 권명준, 전해성

I. 개발 목표

배포된 pintos 위에서 userprogram이 실행되고 종료되는 것을 수행할 수 있도록 한다. 또한 userprogram이 system call을 호출하면 그에 맞는 기능을 수행할 수 있도록 한다. 이 때, 프로그램의 안정성을 보장할 수 있도록 한다.

II. 개발 범위 및 내용

가. 개발 범위

process의 실행, 종료 등 다양한 process 수행 시나리오에 대응할 수 있도록 미리 배포된 코드를 수정한다. system call의 구현은 호출 시 넘어온 인자를 user stack에 저장하는 부분부터 실행 후 종료되는 부분까지 하도록 한다. 이번 프로젝트에서 구현해야 하는 system call은 halt, read, write, exec, wait, exit가 있으며, custom system call인 pibonacci와 sum_of_four_integers까지 구현하도록 한다. 이러한 수행들은 그 과정이 콘솔 상에 출력되도록 해준다.

나. 개발 내용

System call의 호출 시 받는 인자를 user stack에 넣는 Argument passing은 적절한 해석을 거친 뒤에 80x86 Calling Convention에 맞추어 유저 메모리에 배치하도록 한다. userprogram이 호출한 system call에 대한 정보는 미리 만들어진 syscall_init 함수를 통해 syscall_handler에 전달된다. handler에서는 이 정보를 해석하고 해당 system call을 호출하게 된다. 각 system call의 기능은 다음과 같으며, pintos의 documentation을 참고하였다.

void **halt** (void)

shutdown_poser_off()를 호출하여 pintos를 종료한다.

```
void exit (int status)
```

현재 실행중인 user program을 종료하고 콘솔 상에 종료된 프로세스의 이름과 status를 출력해준다. 이 때, status를 커널에 알린다. 아래에 서술될 wait함수로 현재 프로세스의 parent가 현재 프로세스의 종료를 기다리고 있다면, parent에도 status를 전달하도록 한다. 전달되는 status가 0이 아닐 경우 비정상적으로 종료된 것이다.

```
pid_t exec (const char* cmd_line)
```

인수로 넘어온 cmd_line에는 실행해야할 실행파일의 이름과 인자가 저장되어 있다. 새로운 프로세스를 만들어 이 실행파일을 메모리에 올린 뒤 실행한 뒤에 해당 프로세스의 id를 반환한다. 프로세스를 실행할 때 새로 만든 프로세스를 현재 프로세스의 child로 저장해주어야 하며 현재 프로세스는 새로 만들 프로세스의 load가 끝날 때까지 끝나선 안 된다. 문제가 있어서 실행이 불가능한 경우 -1을 반환하게 된다.

```
int wait (tid_t child_tid)
```

현재 스레드가 인자로 넘어온 child_tid를 id로 가지는 스레드가 종료되는 것을 가디하게 한다. 종료되었다면 기다린 스레드의 종료 상태를 반환하도록 한다. 다만 child_tid가 현재 스레드의 child가 아니거나, child_tid가 유효하지 않다면 바로 -1을 반환하도록 한다.

```
int read (int fd, void* buffer, unsigned size)
```

file descriptor로 넘어온 fd에서 데이터를 size만큼 읽은 뒤에 buffer에 저장한다. 본 프로젝트에서는 fd가 0인 경우, 즉 표준 입력을 통해 데이터가 들어올 때만 작동할 수 있다.

```
int write (int fd, const void* buffer, unsigned size)
```

file descriptor로 넘어온 fd에 buffer에 있는 데이터를 size만큼 출력하도록 한다. 본 프로젝트에서는 fd가 1인 경우, 즉 표준 출력에만 출력이 가능하다.

아래는 따로 추가되는 custom system call이다.

```
int pibonacci (int n)
```

n번째 피보나치 수열을 반환한다.

```
int sum_of_four_integers (int a, int b, int c, int d)
```

인자로 들어온 네 정수의 합을 반환한다.

III. 추진 일정 및 개발 방법

가. 추진 일정

- 9월 30일 토요일 : 이번 프로젝트에 대한 강의를 듣고 무엇을 해야 하는지 파악한다.
- 10월 1일 ~ 10월 6일 : Pintos manual을 읽으며 공부한다.
- 10월 7일 ~ 10월 8일 : Argument passing을 구현한다.
- 10월 9일 ~ 10월 18일 : Custom system call을 제외한 system call을 구현한다.
- 10월 19일 ~ 10월 20일 : Custom system call 구현 및 보고서 작성

나. 개발 방법

배포된 pintos의 소스코드를 먼저 분석하여 pintos가 실행 흐름이 어떻게 되는지를 알아낸다. 이 때, 소스코드 분석을 위해 ctags와 cscope를 사용한다. 분석 이후 실제 구현을 위해 수정해야 할 파일을 추려낸다. 그 후에 각 연구원끼리 구현할 컴포넌트를 구분한 뒤에 개발을 시작하도록 한다. 소스코드를 분석 한 결과 주어진 코드는 userprogram이 system call이 호출 되었을 때, interrupt가 걸리고 그에 맞춘 interrupt handler가 구현되어 있는 것을 확인할 수 있었다. 또한 실행파일을 열고 메모리에 올리는 부분도 구현되어 있었기 때문에 실제로 구현해줘야 할 부분은 system call의 핸들러 와 각 system call을 실행하기 위해 argument를 메모리에 올리는 부분과 실행하는 부분이었다. 이를 아래와 같이 분담하여 프로젝트를 진행하였다. 또한 user memory access가 일어날 때마다 그 주소가 valid한지 체크하는 방식으로 개발하도록 한다.

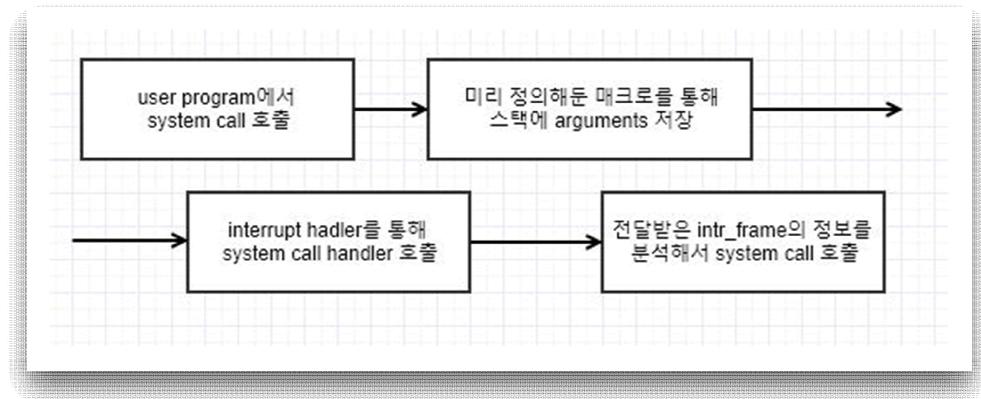
다. 연구원 역할 분담

- 권명준 : Argument passing 및 Custom System Call 구현
- 전해성 : System Call 구현

IV. 연구 결과

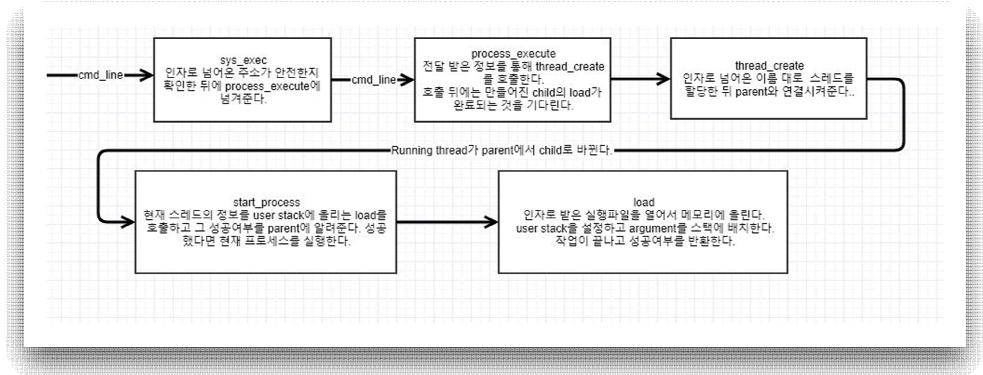
1. 합성 내용

전체 프로그램의 구성은 다음과 같다.

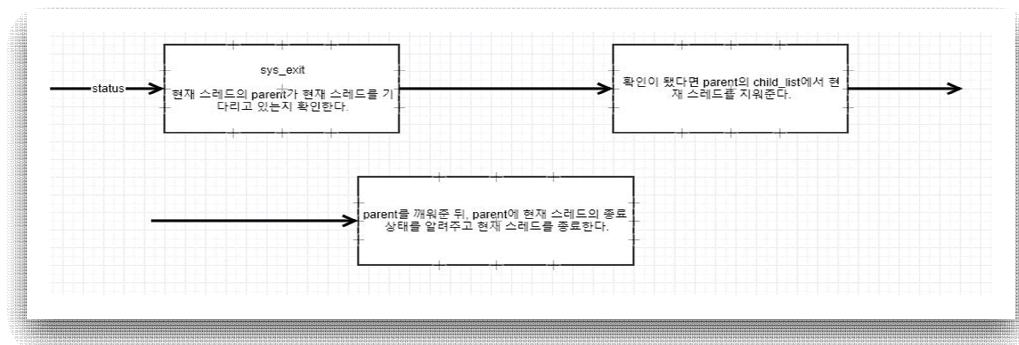


각 system call의 순서도는 아래와 같다.

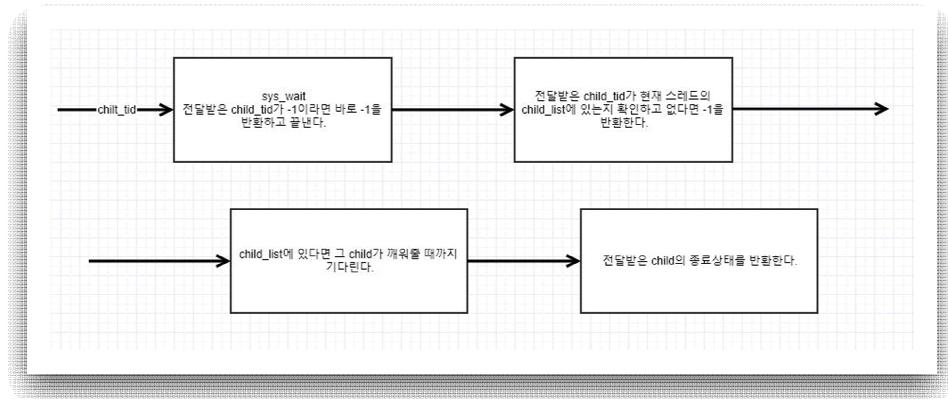
sys_exec



sys_exit



sys_wait



2. 제작 내용

사용한 자료구조

struct thread

기존 코드에 있던 자료구조에 개발 목표에 맞춰 마땅한 변수 몇 가지를 추가하였다. 추가한 변수에 대한 상세내역은 다음과 같다.

struct therad *parent

현재 스레드의 parent를 가리키는 변수다. sys_wait, exit를 구현하기 위해서 추가하였다.

tid_t cur_child

현재 스레드가 기다리고 있는 child process의 id이다. sys_exit를 구현하기 위해서 필요한 변수이다.

bool child_load_successful

새로 만들어진 child process의 load작업이 제대로 끝났는지 알기 위해 만든 변수이다.

int child_status

현재 기다리고 있는 child의 종료 상태를 저장하기 위한 변수이다.

struct semaphore sema

다양한 프로세스 실행 시나리오에 대응하는 동기화 작업을 위해 추가한 변수이다.

struct list child_list

현재 스레드와 연결된 child_process의 리스트이다.

struct child_process

child process를 좀 더 관리하기 쉽도록 하기 위해 추가한 자료구조로 struct thread의 child_list의 원소이다.

struct list_elem elem

child_list를 쓰기 위한 변수이다.

tid_t tid

현재 들고 있는 child process의 tid이다.

struct thread *child

만들어진 child process를 가지는 변수이다.

위 자료구조들 외에도 interrupt의 정보를 가지고 있는 intr_frame도 sys_exec와 handler에서 사용한다.

알고리즘

argument passing

process_execute 함수를 살펴보면 file_name 문자열을 복사한 다음 thread_create()를 호출한다. thread_create()는 새 thread를 만드는데 이때 name은 file_name, function은 start_process, 이의 인수는 file_name을 복사한 fn_copy가 된다. start_process()가 받은 인수 fn_copy는 복사본이므로 마음대로 수정해도 문제가 없음을 알 수 있다. start_process()는 load()를 호출한다. 따라서 load()를 수정해서 argument passing을 구현하는 것이 적절하다고 판단했다.

load 안에서 file_name 문자열을 strtok_r()을 사용하여 공백 단위로 분해한다. 파라미터의 개수를 argc에 저장하고 각각 분해된 문자열을 argv[i]에 저장한다. argv는 char**가 된다.

그 다음은 load()를 수정해 인수들을 80x86 calling convention에 맞게 스택에 저장한다. 맨 위에 argv들의 데이터가 들어간다. 이때 어차피 포인터로 처리할 것이라 순서는 상관이 없지만 convention에 맞게 오른쪽에서 왼쪽 규칙에 따랐다. 그리고 *esp를 감소시키면서 차례대로 word-align 값, argv[i]들, argv, argc, return address를 저장했다. 이때 이 함수는 반환하지 않으므로 fake address가 들어가므로 0을 사용했다.

System calls

user program이 system call을 호출하게 되면 pintos에 미리 정의된 매크로를 통해 인자들을 intr_frame에 저장시킨 뒤에 system call에 대한 interrupt를 통해 syscall_handler가 호출되게 되는데 미리 정의된 매크로 중에선 인자가 4개인 매크로가 없다. 추가될 custom system call중에 sum_of_four_integers는 인자가 4개이기 때문에 새로운 매크로 하나를 추가해준다. syscall_handler가 호출되면 넘어온 intr_frame의 스택포인터가 유효한지 먼저 체크해준다. 이후에도 user한테서 넘어온 모든 주소는 접근 전에 그 주소가 유효한지 체크하고 그렇지 않다면 바로 종료되도록 한다.

syscall_handler에서는 넘어온 interrupt의 정보를 해석하여 어떤 system call을 호출해야 할지 정한 뒤에 만들어진 스택에서 argument를 가져와 system call을 호출해준다.

sys_exec는 시스템 콜 exec를 수행하는 함수이며, 넘어온 실행파일을 실행하는 child process를 만들어 준다. 현재 스레드는 child process를 만든 뒤에 child process의 load작업이 완료되는 것을 기다리기 위해서 재운다. 이는 현재 스레드의 semaphore를 이용해서 수행된다. child process가 만들어지고 실행되는 함수는 각각 thread_create와 start_process로 create에서 child의 parent를 연결시켜주고, start_process에서 parent의 child_list에 현재 만들어진 child process를 추가해준 뒤에 load를 통해 실행파일을 메모리에 옮려주고, 인자들을 user stack에 옮겨준다. load가 완료된 뒤에 load결과를 parent에 전달하고, parent를 깨워준다. load가 실패한다면 만든 child를 종료시킨다. parent는 child가 깨워준다면 child의 tid를 반환한다.

sys_exit는 시스템 콜 exit를 수행하는 함수이며, 이 함수를 통해 종료되는 스레드의 이름과 종료 상태를 출력해준다. process를 종료시킬 때, parent가 현재 스레드를 기다리고 있는지 확인하며 현재 스레드를 기다릴 때까지, thread_yield함수를 이용하여 현재 스레드가 계속 running queue에 머물게 한다. parent가 현재 스레드를 기다리고 있는

것이 확인되면, parent의 child_list에서 현재 스레드를 삭제해주고, 종료 상태를 전달해준다. 그 뒤에 parent의 semaphore를 통해 깨워주고 현재 스레드를 종료해준다.

sys_wait는 시스템 콜 wait를 수행하는 함수이며, 인자로 넘어온 pid가 종료되는 것을 기다리는 것이 주된 기능이다. 실행이 되면 제일 먼저 pid가 -1인지 확인하고 -1이라면 -1을 반환하며 함수를 바로 종료한다. 현재 스레드의 child_list를 탐색하며 child의 tid와 pid가 같은 child를 찾는다. 해당하는 child가 없다면 -1을 반환하고 종료한다. 해당하는 child를 찾았다면 자신이 기다리고 있는 cur_child를 pid로 바꾸고 자신의 semaphore를 내리고 그 child가 자신을 깨워주는 것을 기다린다. child가 깨워준 뒤에는 cur_child의 종료 상태를 반환한다.

sys_read, sys_write의 경우 인자로 전달받은 file descriptor에 대하여 읽기, 쓰기 작업을 수행하는 함수이다. 본 프로젝트에 대해서는 각각 표준 입력, 표준 출력에 대해서만 작동하게 된다. 인자로 넘어온 buffer가 valid한지 확인하고 invalid하면 현재 스레드를 종료시킨다. 입력받는 데에는 pintos내부의 함수 input_getc를 사용하였고, 출력에는 putbuf를 사용하였다.

sys_halt는 pintos를 종료해주며, pintos에 미리 정의된 함수 shutdown_power_off를 호출한다.

pibonacci, sum_of_four_integers는 미리 정의된 기능을 수행해준다. 이를 수행하기 위해 lib/userprog/syscall.c에 syscall4를 추가로 정의하고, lib/syscall-nr.h에 해당 시스템콜에 대한 syscall number를 추가했다. syscall4는 단순한 매크로로 인자가 4개인 시스템 콜에 대한 스택 조작을 해준다. 이는 sum_of_four_integers를 수행하기 위해 추가되었다.

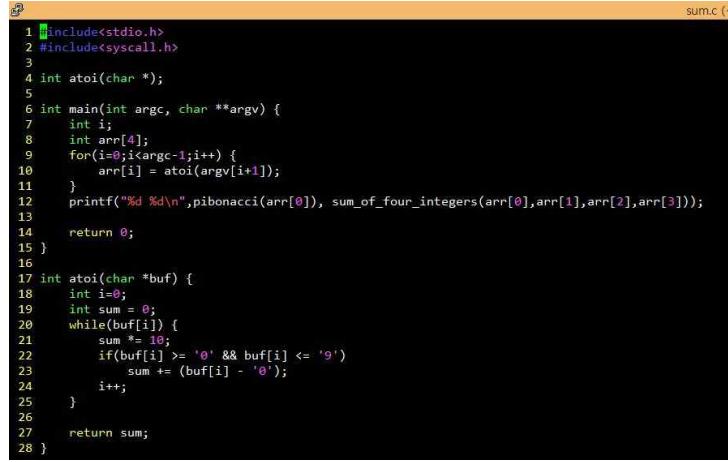
3. 시험 및 평가 내용:

기본적인 system call들(custom system call들을 제외한)에 대한 평가는 배포된 pintos 내부에 존재하는 테스트들로 이루어 지며 본 프로젝트에서 통과하여야 하는 테스트는 총 21개로 아래는 그 테스트들을 수행한 뒤 통과한 것을 찍은 것이다.

```
pass tests/userprog/args-none  
pass tests/userprog/args-single  
pass tests/userprog/args-multiple  
pass tests/userprog/args-many  
pass tests/userprog/args-dbl-space  
pass tests/userprog/sc-bad-sp  
pass tests/userprog/sc-bad-arg  
pass tests/userprog/sc-boundary  
pass tests/userprog/sc-boundary-2  
pass tests/userprog/halt  
pass tests/userprog/exit
```

```
pass tests/userprog/exec-once  
pass tests/userprog/exec-arg  
pass tests/userprog/exec-multiple  
pass tests/userprog/exec-missing  
pass tests/userprog/exec-bad-ptr  
pass tests/userprog/wait-simple  
pass tests/userprog/wait-twice  
pass tests/userprog/wait-killed  
pass tests/userprog/wait-bad-pid  
pass tests/userprog/multi-recuse
```

Custom system call을 평가하기 위한 테스트 프로그램은 직접 만들어서 수행했으며, 아래와 같이 작성하였다.

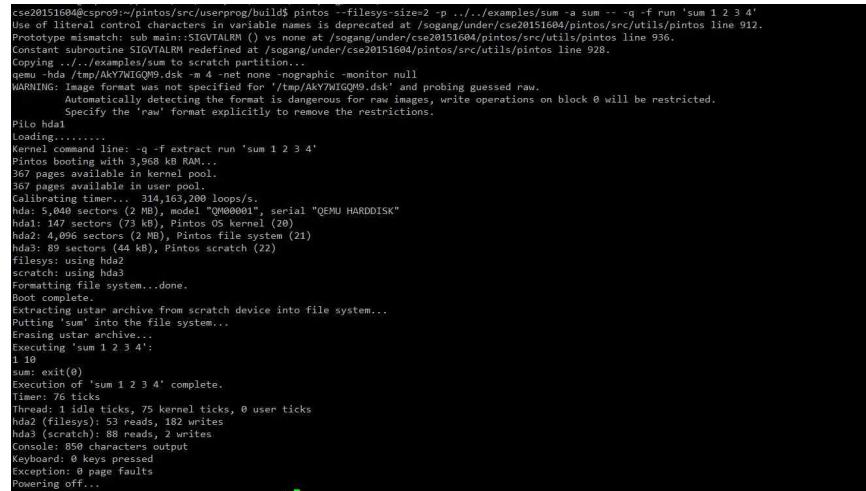


```

1 #include<stdio.h>
2 #include<syscall.h>
3
4 int atoi(char *);
5
6 int main(int argc, char **argv) {
7     int i;
8     int arr[4];
9     for(i=0;i<argc-1;i++) {
10         arr[i] = atoi(argv[i+1]);
11     }
12     printf("%d %d\n",fibonacci(arr[0]), sum_of_four_integers(arr[0],arr[1],arr[2],arr[3]));
13
14     return 0;
15 }
16
17 int atoi(char *buf) {
18     int i=0;
19     int sum = 0;
20     while(buf[i]) {
21         sum *= 10;
22         if(buf[i] >= '0' && buf[i] <= '9')
23             sum += (buf[i] - '0');
24         i++;
25     }
26
27     return sum;
28 }

```

아래는 이 코드로 만들어진 실행파일을 pintos 위에서 실행한 결과 화면이다. 실행파일의 이름은 sum이다.



```

cse20151604@cspro9:~/pintos/src/userprog/build$ pintos -f filesystem-size=2 -p ../../examples/sum -a sum -- -q -f run 'sum 1 2 3 4'
Use of illegal control characters in variable names is deprecated at /sogang/under/cse20151604/pintos/src/utils/pintos line 912.
Prototype mismatch: sub main::SIGVTALRM () vs none at /sogang/under/cse20151604/pintos/src/utils/pintos line 936.
Constant subroutine SIGVTALRM redefined at /sogang/under/cse20151604/pintos/src/utils/pintos line 928.
Copying ./examples/sum to scratch partition...
qemu -hda /tmp/AAKY7WIGQ9.dsk -m 4 -nographic -monitor null
WARNING: Image format was not specified for '/tmp/AAKY7WIGQ9.dsk' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.
Pilo hdal
Loading.....
Kernel command line: -q -f extract run 'sum 1 2 3 4'
Pintos booting with 3,968 kB RAM...
367 pages available in user pool.
367 pages available in user pool.
Calibrating timer... 314,163,200 loops/s.
hda: 5,040 sectors (2 MB), model "QEMU HARDDISK"
hda1: 147 sectors (73 kB), Pintos OS kernel (28)
hda2: 4,896 sectors (2 MB), Pintos file system (21)
hda3: 89 sectors (44 kB), Pintos scratch (22)
filesys: using hda2
scratch: using hda3
Formatting file system...done.
Root complete.
Extracting ustar archive from scratch device into file system...
Putting 'sum' into the file system...
Erasing ustar archive...
Executing 'sum 1 2 3 4':
1 10
sum: exit(0)
Execution of 'sum 1 2 3 4' complete.
Idle ticks
Thread: 1 idle ticks, 75 kernel ticks, 0 user ticks
hda2 (filesys): 53 reads, 182 writes
hda3 (scratch): 88 reads, 2 writes
Console: 850 characters output
Keyboard: 0 keys pressed
Exception: 0 page faults
Powering off...

```

V. 기타

1. 연구 조원 기여도: 권명준(50%) 전해성(50%)

2. 기타 본 설계 프로젝트를 수행하면서 느낀 점을 요약하여 기술하라. 내용은 어떤 것 이든 상관이 없으며, 본 프로젝트에 대한 문제점 제시 및 제안을 포함하여 자유롭게 기술할 것.

(느낀 점)

- 권명준: Parameter passing은 어셈블리 시간에 배웠던 esp와 ebp를 사용하여 매개변수를 넘겨주고 이를 해석하고 반환 주소를 저장해놓는 일련의 과정을 실제로 구현하는 느낌이었다. 상당히 추상적으로 느껴졌고 assembly instruction으로만 보던 것을 이제 C로 직접 구현하니 생각보다 어려웠다. 가장 어려웠던 부분은 void**인 esp의 형식 문제였다. argv의 주소를 저장할 때 $\ast(\text{void} \ast \ast)(\ast \text{esp}) = (\text{char} \ast \text{값})$ 으로 형 변환 시켜줘야 주소 값이 제대로 들어간다. 형 변환을 잘못하면 esp에 이상한 값이 쓰여 kernel panic이 떠버려 고생했다. parameter passing도 문제였지만 system call은 더욱 문제였다. process와 thread가 pintos에서 어떻게 처리되는지 manual과 실제 코드를 보면 파악하는 데에 상당한 시간이 소요되었다. 자세한 수행 과정이 manual에 없다는 점이 아쉽다. 조금 더 친절하게 호출 경로를 적어줬으면 좋았을 것이다. 또한 thread가 구조체라는 것이 신선했다. 뭔가 새로운 형태의 자료구조가 있을 줄 알았는데 계속 써오던 구조체로 구현이 된다는 게 신기했다. 또한 semaphore는 처음 접하는 개념인데 단순한 원리로 synchronization이 구현되는 것이 신기했다. 여기에서 파트너 전해성 학우의 도움을 많이 받았다. 처음 이 프로젝트 강의를 들을 때에는 도대체 뭘 하라는 건지 말이 안 되고 할 수나 있을까 싶었는데 결국 해낼 수 있어서 기쁘다.

- 전해성 : 수업시간에 배운 process의 수행 시나리오와 동기화 과정을 본 프로젝트를 통해 구현해봄으로써 교과목에 대한 이해가 더 깊어지는 것을 알 수 있었다. 프로젝트를 처음 시작할 때, 정말 뭐부터 손대야 할지 감도 안 와서 상당히 고통스러웠다. 차분하게 소스코드들을 분석하면서 pintos의 실행과정을 알고 나니 프로젝트의 진행이 가능했다. 본 실습은 대부분의 수강생이 학교 측에서 제공하는 서버 위에서 진행하게 되는데 분석할 때, ctags만으론 조금 부족한 느낌이 있었다. 그래서 제공받은 서버가 아닌 개인 서버를 이용해서 추가적으로 cscope를 추가로 사용하였다. 서버에 cscope도 깔아주면 좋겠다. 그리고 이번 학기는 연휴 때문에 늦어져서인지 동기화에 대한 수업이 프로젝트 마감일이 있는 주에야 시작되었다. 이 점 또한 아쉬웠다.