**Pintos Project 1 : User Program (1)**

**(설계 프로젝트 수행 결과)**

과목 명 : [CSE4070] 운영체제

담당 교수 : 김영재 교수님

조 / 조원 : 50조, 20171672 이정원, 20171698 최아연

개발 기간 : 2019/10/6 – 2019/11/3

**프로젝트 제목 : Pintos Project 1 User Program (1)**

**제출일 : 2019년 11월 3일**

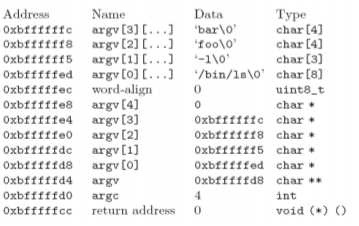
**참여 조원 : 이정원, 최아연**

1. **개발 목표**

* 사용자에 의한 기본 명령에 대해 프로그램의 실행 환경을 구축하는 것이 이번 프로젝트의 목표이다. 개발 전 기존 Pintos에는 System call, System call handler, Argument passing, User stack이 구현되어 있지 않아 Pintos가 사용자 프로그램을 실행할 수 없다.
* Argumemnt passing을 통해 사용자 프로그램의 mutiple arguments를 분석하여 메모리에 할당하는 기능과, 유효하지 않은 포인터를 어떻게 대처할지 처리하는 기능과, 사용자의 명령에 따라 각각 기능을 수행하는 System call handler의 기능과, 추가적인 기능(fibonacci, sum of four int)을 추가할 것이다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **Argument Passing**
* 주어진 코드는 stack page만을 할당하기 때문에, arguments를 위한 스택을 userprog/process.c의 load 함수 내에 setup\_stack() 이후에 만들어야 한다. /bin/ls -l foo bar 를 예를 들자면, 스택은 다음 그림과 같아야 한다.



주어진 argument 들을 분석해 스택의 뒤에서부터 차례대로 argument의 길이만큼 집어넣고, argv와 argc, 그리고 return address를 집어넣는다.

* **User Memory Access**
* 주어진 코드는 유효하지 않은 pointer를 넘겨줄 수 있다. 하지만 유효하지 않은 포인터는 커널과 실행중인 process에 손상을 입힐 수 있으므로, 거절되야 한다.
* 유효하지 않은 pointer를 거절하기 위해선 pointer가 유효한지 판단을 먼저 해야 한다. 만약 pointer가 유효하지 않다면, page fault를 야기할 수 있으므로, 주어진 코드에서 userprog/exception.c에서 page\_fault() 함수를 수정한다.
* pageir\_get\_page(), is\_user\_vaddar(), is\_kernel\_vaddr()함수를 이용해 포인터의 유효성을 확인한다.
* User memory를 벗어나 kernel memory에 침범했을 시, 에러처리를 해야 한다.
* **System Calls**
* 시스템 호출을 위해 halt, exit, exec, wait, read write의 기능을 수행하게 구현한다. 시스템 호출의 결과를 반환하기 위해 userprog/syscall.c의 syscall\_handler() 함수를 구체화한다.
* halt: shutdown\_power\_off()를 호출하여 pintos를 종료한다.

exit: 실행 중인 사용자 프로그램을 종료하고 종료된 프로세스의 이름과 status를 출력해 준다.

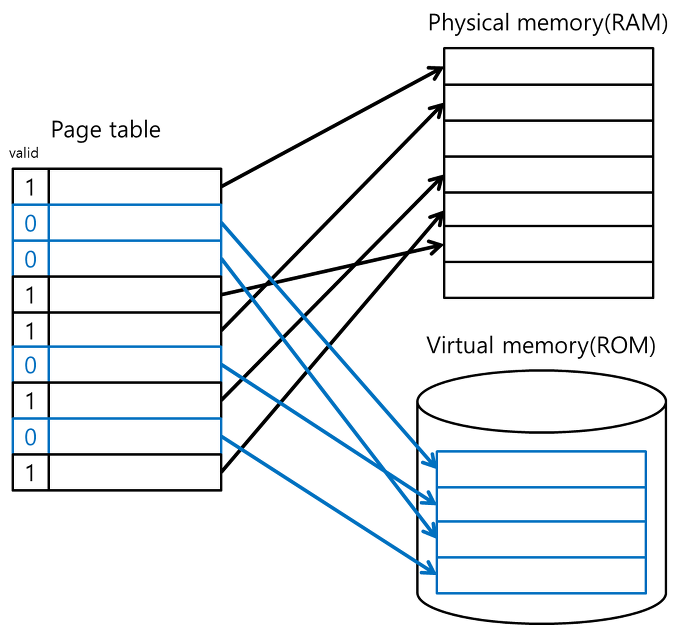
exec: 새로운 프로세스를 만들어 이 실행 파일을 메모리에 올린 뒤 실행한 뒤에 해당 프로세스의 id를 반환한다.

wait: 현제 스레드가 인자로 넘어온 child\_tid를 id로 가지는 스레드가 종료되는 것을 기다리게 한다. 이 wait를 구현하기 위해 userprog/process.c의 process\_wait 함수를 구현하고, 부모와 자식에 대한 정보를 저장하기 위해 thread/thread.h의 thread 구조체에 추가한다.

read, write: input\_getc를 이용해 read(STDIN)에 관한 기능을 구현하고, putbuf를 이용해 write(STDOUT)에 관한 기능을 구현한다.

fibonacci, sum\_of\_four\_int: 이번 프로젝트에서 추가적으로 구현해야 하는 시스템 호출이다. fibonacci는 n번째 피보나치 수열을 반환하고, sum\_of\_four\_int는 입력받은 네 정수의 합을 반환한다.

* 1. **개발 내용**
* **Page & Process**
* 운영체제의 가상 메모리(Virtual Memory)는 RAM을 관리하는 방법 중 하나로, 각 프로그램에 실제 메모리 주소가 아닌 가상의 메모리 주소를 할당하는 방식을 말한다. RAM의 부족한 용량을 보완하는 데 주로 쓰인다. 운영체제는 가동되고 있는 프로세스들의 내용(페이지) 중, 덜 중요한 것들을 하드 디스크의 공간에 옮겨 놓는다. 그리고 프로세스가 동작하는 도중, 메모리에 필요한 데이터(페이지)가 없으면 하드 디스크를 찾아 해당 데이터를 가져온다. 여기서 페이지(Page)란, 가상 메모리를 사용하는 최소 크기 단위이다. 페이지는 프로세스의 구성 요소라고 볼 수 있다.
* 운영체제는 페이지 테이블(Page Table)로 가상 메모리를 관리한다. 페이지 테이블을 확인하여 가상 메모리에 페이지가 있는지 확인하고, 없으면 프로세스를 중단한다.



* **User process가 Kernel memory에 접근하는 것을 막기 위한 방법에 대해 설명**

1. 사용자 제공 포인터의 유효성을 확인한 수 역 참조

* ‘userprog / pagedir.c’및 ‘threads / vaddr.h’의 함수들을 이용하는 방법이다. 사용자 메모리 처리의 가장 간단한 방법이다. 이를 위해서 사용되는 함수는 is\_user\_vaddr(), is\_kernel\_vaddr()이다.

2. 포인터가 PHYS\_BASE 아래를 가리키는지 확인한 후 역참조

- 'userprog / exception.c'의 fage\_fault() 함수를 활용한다. 일반적인 프로세서의 MMU를 활용하기 때문에 실제 커널에서 사용되는 경향이 있다.

- 두 경우 모두 리소스를 leak하지 말아야 한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* 2019년 10월 28일 – 2019년 10월 29일: 코드분석 및 구현해야 할 항목 파악
* 2019년 10월 30일: argument passing, user memory access구현
* 2019년 10월 31일: system-call 구현
* 2019년 11월 1일: 추가구현(sum, Fibonacci 구현)
* 2019년 11월 2일: wait()오류 수정 및 보고서 작성
* 2019년 11월 3일: 보고서 작성
  1. **개발 방법**
* **Argument Passing**
* userprog/process.c load()함수를 수정한다.
* 우선 stack이 set up되어 있어야한다.
* 주어진 argument 들을 strtok()를 활용해, token화 한 후 스택의 뒤에서부터 차례대로 argument의 길이만큼 집어넣는다. 그 다음, argv와 argc, 그리고 return address를 집어넣는다
* 구현이 잘 되었는지 확인하기 위해, hex\_dump()함수를 활용한다.
* **User Memory Access**
* -userprog/exception.c에 page\_fault()함수를 수정한다.
* Is\_user\_vaddr() 함수를 호출해 커널의 virtual adress를 return받는다. 이때, UADDR이 unmapped되어 있으면, null pointer를 return하는데, 이때 syscall\_exit()를 호출에 에러처리를 한다.
* 또한 pagedir\_get\_page()함수를 통해서도 에러처리를 한다.

* **System Calls**
* userprog/syscall.c의 syscall\_handler() 함수의 정상적인 수행을 위해 switch를 이용하여 각각의 시스템 호출에 대응하여 기능들을 구현한다. 해당 시스템 호출하기 이전에, 주어진 주소가 NULL이거나, kernel virtual address인지 확인하는 함수를 추가적으로 구현하여 부른다.
* syscall\_halt(): shutdown\_power\_off() 호출

syscall\_exec(cmd\_line): process\_execute(cmd\_line) 호출

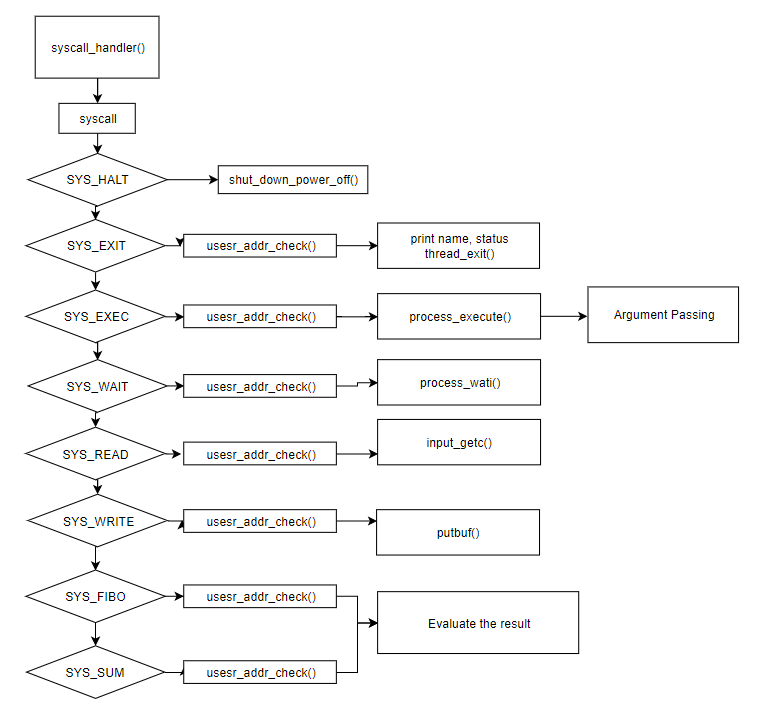
syscall\_wait(pid): process\_wait(pid) 호출. userprog/process.c의 process\_wait 함수를 구현한다. 주어진 child\_tid가 유효한지 확인하고, child\_thread를 찾아 parent를 업데이트한다. 특히, 해당 child\_thread의 status를 확인하고 busy waiting 테크닉을 구현하기 위해 device/timer.c의 timer\_sleep 함수를 호출한다.

syscall\_read, syscall\_write: 입력받은 fd가 유효한지 확인하고 input\_getc(), putbuf()를 이용하여 입출력 기능을 구현한다.

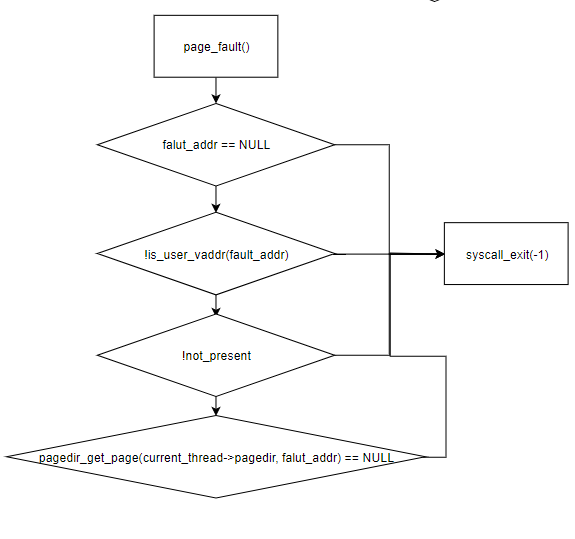
* 1. **연구원 역할 분담**
* 이정원: Argument passing, System call 구현, 보고서 작성
* 최아연: Additional system call, User memory access 구현, 보고서 작성

1. **연구 결과**
   1. **합성 내용**

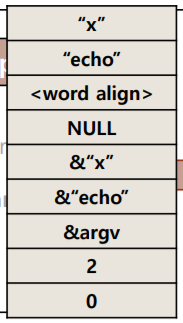
* **syscall\_handler()**

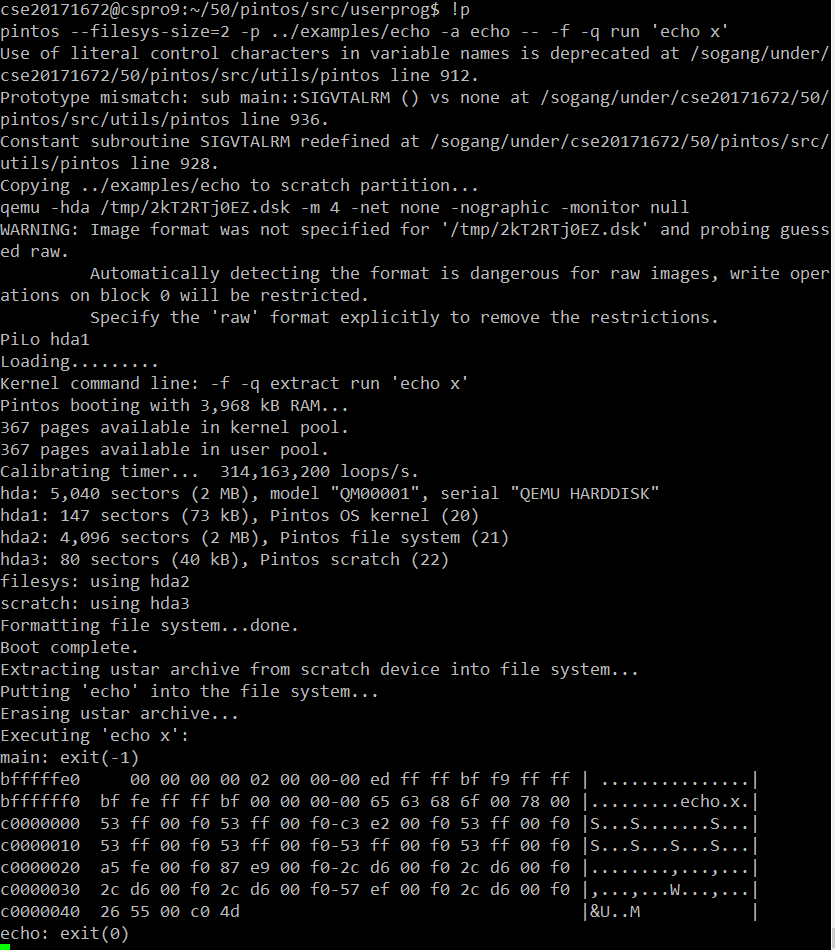


* **userprog/exception.c의 page\_fault()**



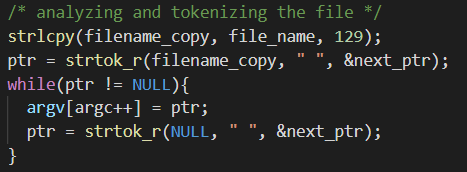
* 1. **제작 내용**
* **Argument Passing**



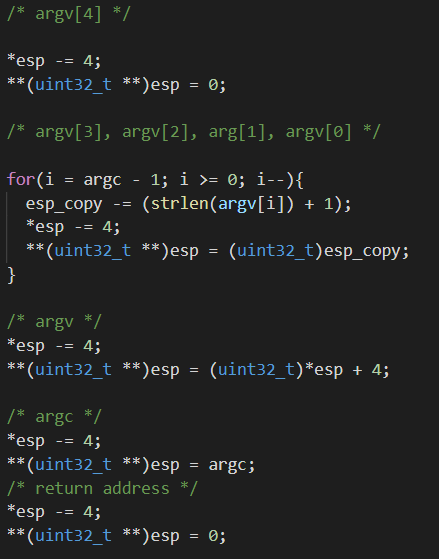


hex\_dump를 이용하여 argument passing 확인

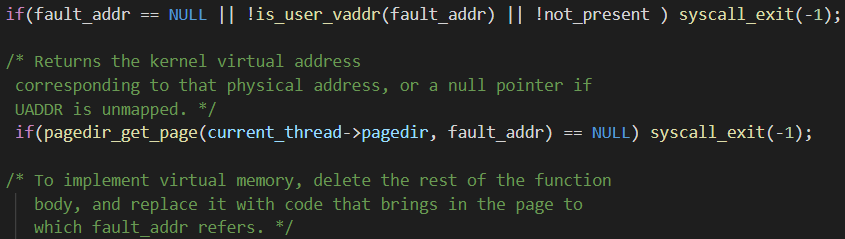
* + - userprog/process.c의 load
* 명령어를 strtok()를 활용해 토큰화한다.



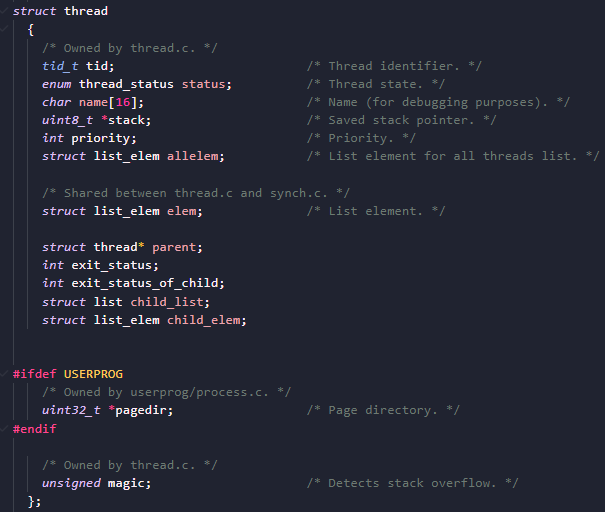
* Word-align을 for문을 통해 실행한다.
* Esp를 복사한 후 스택에 값을 집어넣을 때 활용한다. 스택의 뒤에서부터 차례대로 argv를 입력한다. 그리고 난 후 argc와 return adress도 입력한다.



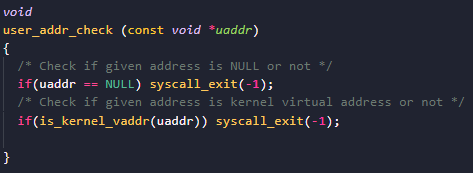
* **User Memory Access**
* is\_user\_vaddar()로 fault\_addr의 값이 null인지 확인한다.
* Pagedir\_get\_page를 통해서도 예외 처리를 한다.



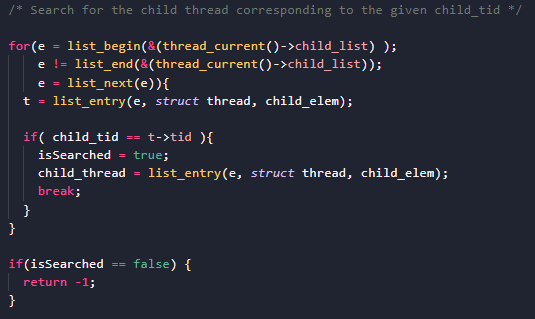
* **System Calls**
* 사용한 Thread 구조체



* list 구조체를 이용하여 자식 스레드 항목과, 해당 스레드의 부모 정보인 parent 스레드 구조체, 그리고 exit status와 exit status of child을 저장하는 int형 변수를 추가해 주었다.
* userprog/syscall.c의 void user\_addr\_check(const void \*uaddr)

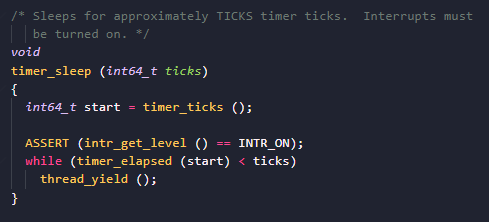


* 시스템 호출하기 전, 주어진 주소가 유효한지 체크하는 함수를 추가로 구현하였다. NULL이거나, is\_kernel\_vaddr을 이용하여 접근 가능한 주소인지 체크하여 유효하지 않은 경우 syscall\_exit(-1)를 호출하여 종료된다.
* userprog/syscall.c의 void syscall\_halt(void)
* shutdown\_power\_off()를 호출하여 (devices/shutdown.h에 구현되어 있음) Pintos를 terminate한다.
* userprog/syscall.c의 void syscall\_wait(pid\_t pid)
* syscall\_wait에서는 process.c의 process\_wait 함수를 호출한다.
* userprog/process.c의 int process\_wait(tid\_t child\_tid UNUSED)
* process\_wait 함수에서는 입력받은 child\_tid를 이용해 child를 찾고, 현재 스레드의 exit code를 반환하는 기능을 한다. 우선 child\_tid가 유효한지 확인하여 유효하지 않으면 바로 -1을 반환한다. 테스트 케이스의 ‘wait-twice’를 위하여 전역 변수 wait\_count를 이용해 wait가 몇 번 실행되었는지 확인하여 2번 이상이라면 바로 -1을 반환한다. child\_thread를 찾기 위해서 이전 프로젝트에서 사용했던 방법을 사용한다. for 문을 이용하여 child\_thread를 찾는다. 찾지 못하면 -1을 반환한다. 해당 코드 부분은 다음과 같다.

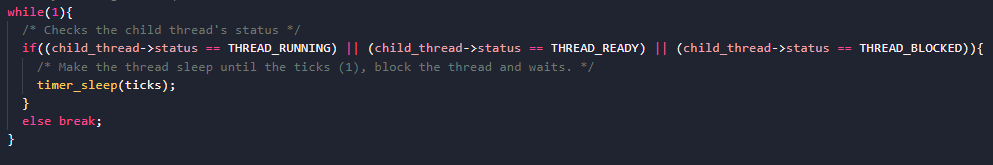


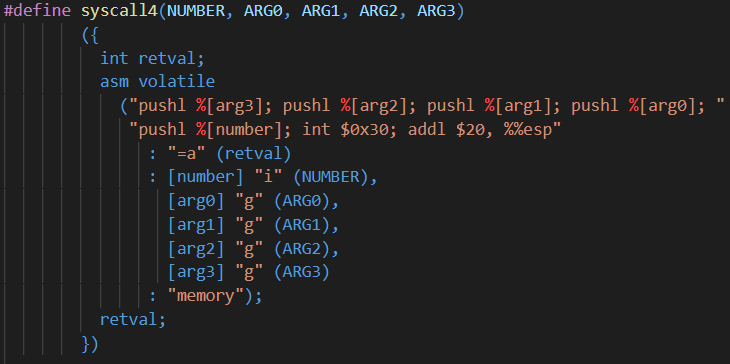
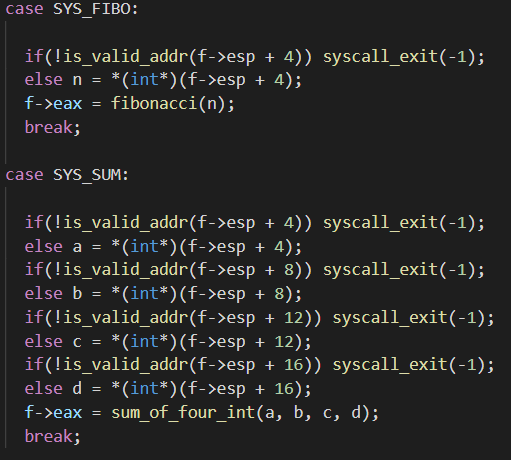
child\_thread의 부모 정보를 업데이트하고, wait\_count를 업데이트한다.

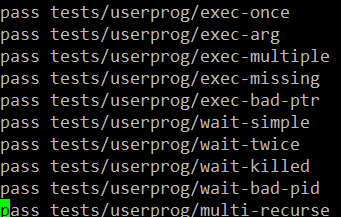
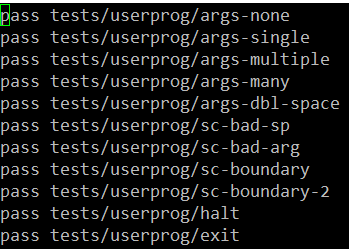
* exit status를 반환하기 이전에 프로세스의 termination을 방지하기 위해, busy waiting 기법을 사용한다.
* Busy Waiting
* 프로세스를 재울 때 시스템 자원 낭비를 최소화하는 방법이다. device/timer.c의 timer\_sleep 함수는 현재 스레드를 ticks 시간 동안 잠재우는 함수이다.



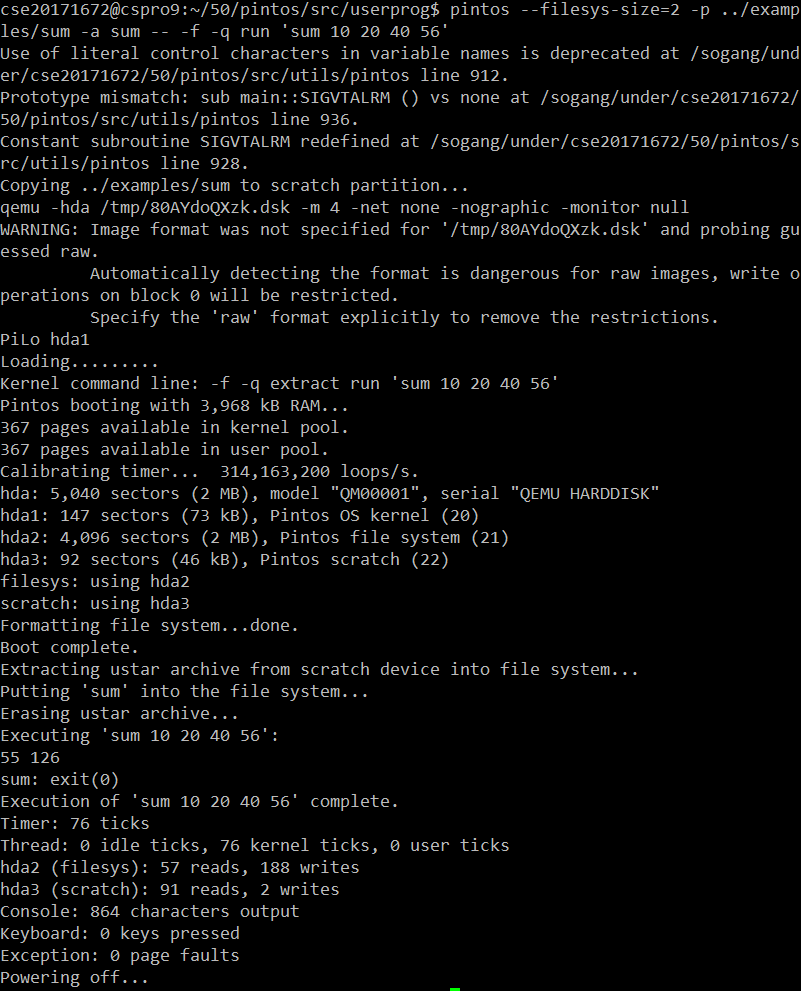
* 이 함수가 호출된 스레드는 while문 내부에 진입하고, 스케줄링에 의해 자신의 차례가 올 때마다 코드가 다시 실행되고 timer\_elapsed(start) 함수를 호출한다. 이 함수의 반환값이 timer\_sleep의 인자인 ticks값보다 작으면 thread\_yield()를 호출하여 read list에 있는 다른 스레드를 위해 CPU 점유를 반환하고 ready list 가장 뒤로 이동한다. 이 과정은 ticks 동안 반복된다.
* Busy waiting을 위하여 해당 스레드의 status를 확인한다. status가 RUNNING, READY, BLOCKED일 때만 timer\_sleep 함수를 호출하는 방식으로 구현하였다. 해당 코드는 다음과 같다.



* userprog/syscall.c의 void syscall\_exit(int status)
* 현재 스레드를 thread\_current()를 이용하여 찾고, 현재 스레드의 exit status 정보를 status로 업데이트한다. 그리고 주어진 출력 형식에 맞게 현재 스레드의 이름과 status를 출력한다. 그리고 thread\_exit() 함수를 호출한다.
* userprog/syscall.c의 int syscall\_read(int fd, const void\* buffer, unsigned size)
* 입력받은 fd가 지정된 범위 내에 있는지, 그리고 버퍼의 주소가 유효한지 확인하는 과정을 거친다. fd가 0일 경우, (STDIN) 반복문을 통해 input\_getc()를 이용하여 읽는 과정을 수행한다. 읽는 과정이 끝나면, 바이트의 size를 반환한다. fd가 0이 아니면 -1를 반환한다.
* userprog/syscall.c의 int syscall\_write(int fd, const void \*buffer, unsigned size)
* syscall\_read와 같이 유효성을 확인하고, fd가 1일 경우 putbuf를 이용하여 콘솔에 출력한다. 쓰는 과정이 끝나면 size를 반환한다. fd가 1이 아니면 -1를 반환한다.
* **Additional System Call**
* Lib/syscall-nr.h에 call number를 추가는다. (SYS\_SUM, SYS\_FIBO)
* Lib/user/syscall.c에 Sum\_of\_four\_int()에서 4개의 변수를 사용하므로, syscall4를 syscall1,2,3와 비슷한 방법으로 구현한다.
* 
* Lib/user/syscall.c에 syscall을 활용해 Sum\_of\_four\_int()와 Fibonacci()를 구현한다.
* Userprog/syscall.c에서 syscall\_handler에 sys\_sum과 sys\_fibonacci를 추가한다.
* 
* Userprog/syscall.c에서 Sum\_of\_four\_int()와 Fibonacci()를 실제로 구현한다.
* Sum\_of\_four\_int()함수는 입력받은 네 수를 더한 값을 return해주고, Fibonacci()함수는 for 문을 통해 fibo수를 구하도록 구현하였다.
  1. **시험 및 평가 내용**
* **Test Cases 수행 결과**



* 21개 항목에 대하여 모두 pass
* **fibonacci(), sum\_of\_four\_int()의 수행 결과**

****

1. **기타**
   1. **연구 조원 기여도**

* 이정원: 50%
* 최아연: 50%
  1. **소감**
* Pintos환경 상에서 사용자에 의한 기본 명령에 대해 프로그램의 실행 환경을 구축하는 project를 수행하며, 운영체제의 원리에 대해 공부할 수 있었다. 뿐만 아니라 linux환경에서 project를 수행하였는데, 덕분에 linux환경에 대해서도 공부할 수 있었다.
* process\_wait() 함수를 구현하는데 많은 어려움이 있었다. Busy waiting 테크닉을 이해하는데도 시간이 오래 소요되었고, 구현하는데도 많은 어려움이 있었다. 하지만 여러 자료들을 찾아보며, 문제점을 분석했고, 문제를 해결할 수 있었다.
* Syscall\_exit에서 status를 잘못 return해주어서 오류가 생긴 것이었는데, 코드를 찬찬히 분석해보고, 여러 자료를 찾아보면서 문제를 해결할 수 있었다.
* 또한 팀프로젝트였기 때문에, 가독성 좋은 코드를 짜는 것이 중요했다. 평소 나만 알아볼 수 있게 코드를 가독성 좋지 않게 짜는 편이었는데, 이 기회를 통해 개발자에 맞는 코딩 스타일을 익히게 되었다.