**Pintos Project 2 : User Program (2)**

**(설계 프로젝트 수행 결과)**

과목 명 : [CSE4070] 운영체제

담당 교수 : 박성용 교수님

조 / 조원 : 23조, 20171640, 20171702

개발 기간 : 2019/11/2 ~ 2019/11/17

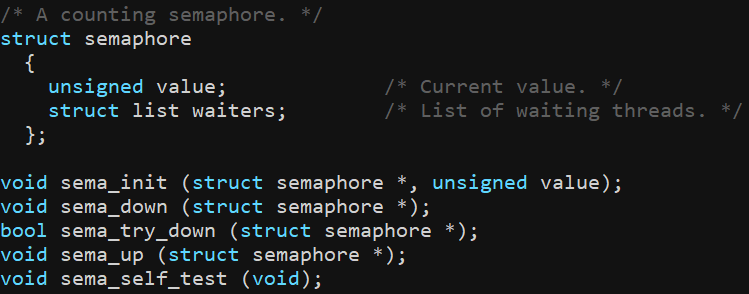
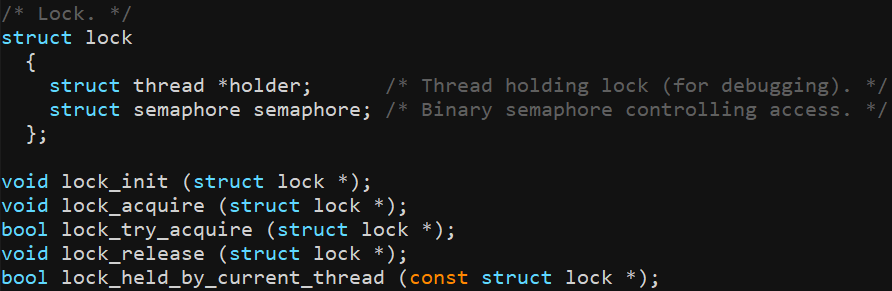
**프로젝트 제목 : Pintos Project 2 User Program (2)**

**제출일 : 2019년 11월 17일**

**참여 조원 : 박수진, 하민주**

1. **개발 목표**

* pintos의 user program에서는 project 1에서 구현한 기본 기능만을 제공하고 file system이 구현되지 않았다. 이번 프로젝트에서는 project 1에서 개발한 내용을 바탕으로 system call 에서 filesys API를 적절히 사용하여 file system관련 함수(create, remove, open, close, filesize, read, write, seek, tell)를 추가적으로 개발한다. 그리고 critical section에 대한 mutual exclusion을 고려하여 synchronization이 적절히 잘 되도록한다. 또한, file\_system function을 사용하여 denying writes to executable files를 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
2. **File system call**  
   **- pintos/src/userprog/syscall.c  
   :** user program에서 파일 시스템을 사용할 수 있도록 system call에 file system function들을 추가한다.  
   추가할 함수 : create(), open(), close(), read(), write(), seek(), tell(), filesize(), remove()
3. **denying writes to executable files   
   - pintos/src/userprog/syscall.c  
   :** running program의 executable file이 수정되지 않도록 해야한다.  
   수정할 함수 : open(), write()
4. **synchronization   
   - pintos/src/userprog/syscall.c  
   :** critical section을 보호하기 위해 file open, read, write할 때 synchronization 해주어야 한다.수정할 함수 : read(), write(), open()  
    **- pintos/src/userprog/process.c  
   :** parent와 child thread가 정상적으로 load 및 exit 되도록 synchronization 해야한다.수정할 함수 : process\_wait(), process\_execute(), process\_exit(), start\_process()
   1. **개발 내용**
      1. **file 관련 system call**filesys API를 사용하여 system call을 구현한다.  
         - create() : filesys\_create()을 이용하여 구현.  
         - remove() : filesys\_remove()을 이용하여 구현.  
         - close() : file\_close()을 이용하여 구현.  
         - open() : file\_open()을 이용하여 구현.  
         - filesize() : file\_length()를 이용하여 구현.  
         - read() : file\_read()를 이용하여 구현.  
         - write() : file\_write()를 이용하여 구현.  
         - seek() : file\_seek()을 이용하여 구현.  
         - tell() : file\_tell()을 이용하여 구현.
      2. **denying writes to executables**Pintos에서는 running program의 executable file이 수정되지 않도록 해야한다.  
         Process들이 write를 하려 하는데 만약 Process가 실행 중 인 상태면  open함수에서 file\_deny\_write()를 통해 제한한다. 그리고 write에서는 deny\_write가 true이면 file\_deny\_write()를 실행하게 하여 제한하였다.
      3. **Synchronization**process 간에 통신을 할 때 공유메모리를 통해 특정 데이터를 공유하게 되는데 이런 경우에는 자원을 동시에 접근할 경우 문제가 발생할 수 도 있다. multi-threaded 프로그램에서는 공유하는 자원에 대해 하나의 데이터에 하나의 프로세스만 접근할 수 있도록 제한해 두어야 하는데 이것이 critical section이다.. 만약 critical section에서 어떤 process가 execute하고 있다면 다른 process는 그 process가 critical section을 빠져나올 때 까지 기다려야한다. 이를 관리하기 위한 기법이 Synchronization(동기화)이다. 이번 프로젝트에서는 2가지의 synchronization 기법이 필요하다. 첫째는 parent와 child thread에 대한 동기화 기법이다. 우선 parent가 여러 개의 child를 가질 때 parent가 모르는 상태로 child가 죽으면 안되기 때문에 parent가 각각의 child에게 wait 명령하기 전까지 죽지 못하도록 해야하고, 또한 child thread가 load되기 전에 parent thread의 execute가 끝나면 안되기 때문에 child의 load 후 parent thread를 execute해야 한다. 또 다른 하나는 file에 접근할 때 하나의 file을 동시에 여러 process가 접근하지 못하도록하는 synchronization이다. pintos의 synchronization API로는 semaphore와 lock이 있다. 첫번째는 semaphore로, 두번째는 lock으로 구현한다.  
           
         1) semaphore  
           
           
         2) lock  
         
5. **추진 일정 및 개발 방법**
   1. **추진 일정**

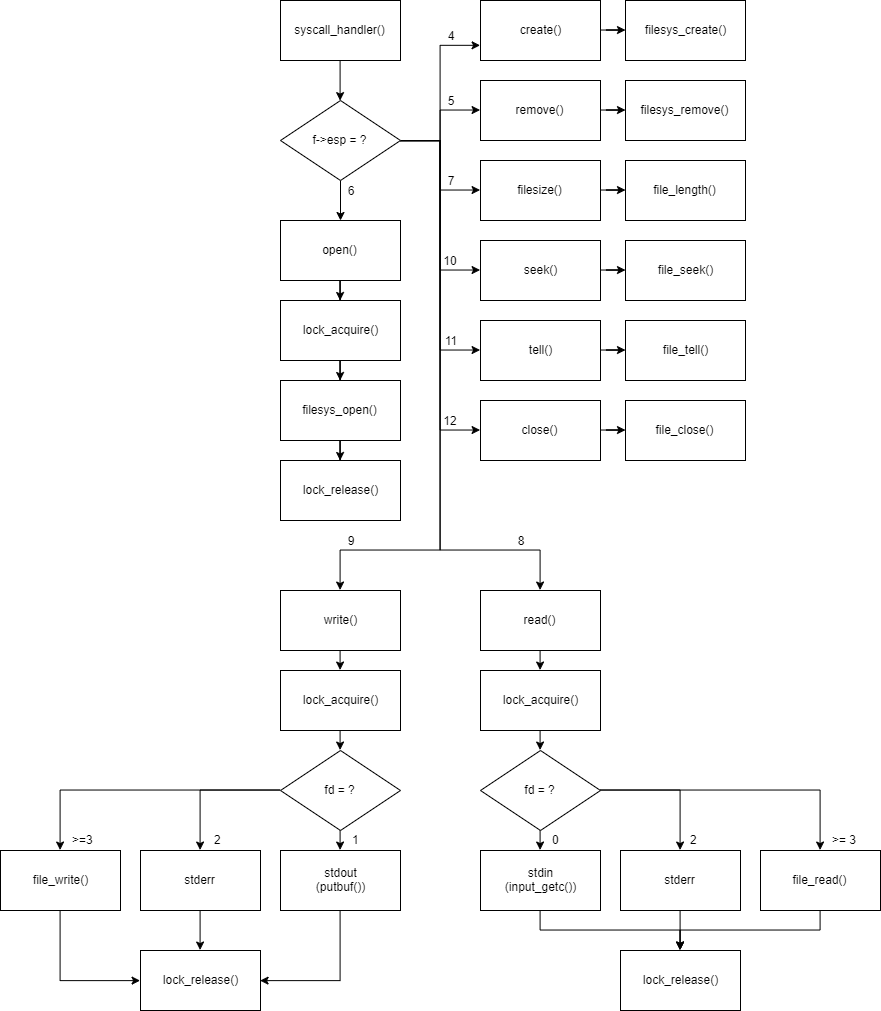
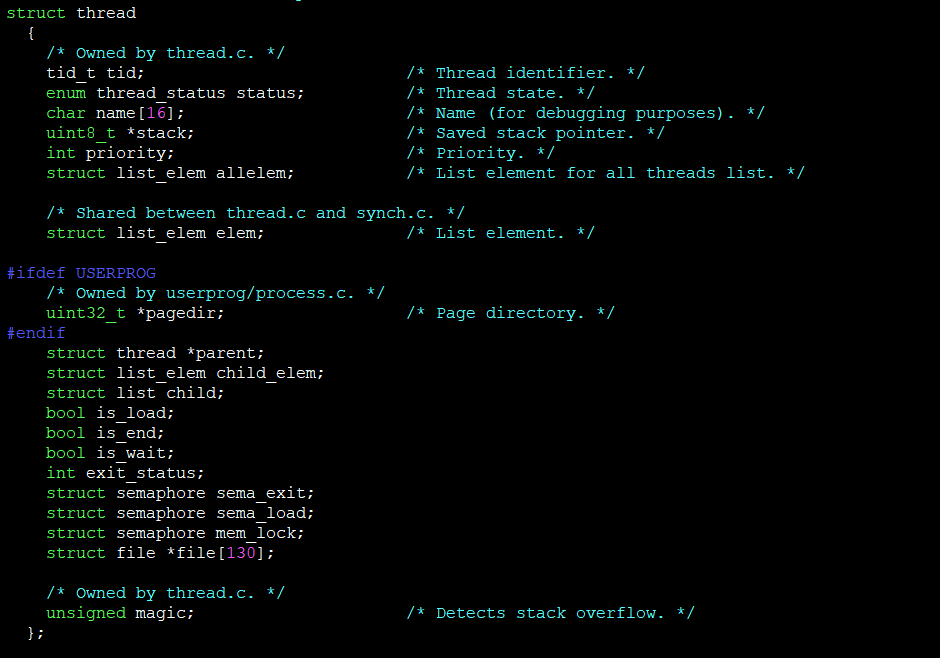
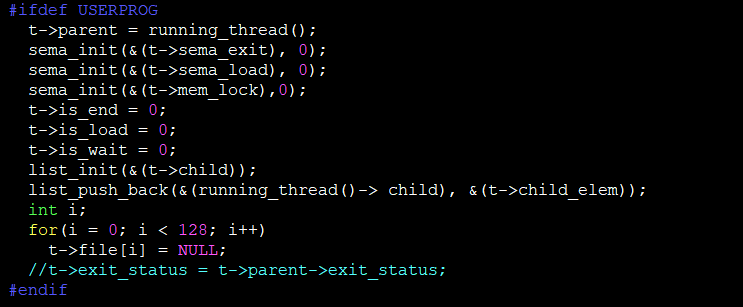
|  |  |
| --- | --- |
| 기간 | 내용 |
| 1주차 (11/2 ~ 11/9) | * 매뉴얼 분석 * 자료구조 및 알고리즘 설계 * file system, denying writes to executable files 구현 |
| 2주차 (11/9 ~ 11/17) | * synchronization 구현 * 테스트 및 코드 수정 * 보고서 작성 |

* 1. **개발 방법**

1. **file system call   
   - pintos/src/userprog/syscall.c  
   :** 파일 시스템을 사용할 수 있도록 file system function들(create(), open(), close(), read(), write(), seek(), tell(), filesize(), remove())을 각각의 function에 해당하는 pintos의 filesys API들을 이용하여 구현하고, system call handler에서 function에 해당하는 system call number를 통해 적절히 호출하여 사용한다.
2. **denying writes to executable files   
   - pintos/src/userprog/syscall.c  
   :** executable file이 수정되지 않도록, open 할때  thread\_current()의 name과 인자로 받은 name이 일치할 경우(현재 실행중인 파일인지 검사) file\_deny\_write()처리를 하여 구현 해줄 것이다.
3. **synchronization   
   - pintos/src/userprog/syscall.c  
   :** file의 cs problem(readers/writers problem)을 해결하기 위해 open, read, write에 lock\_acquire()과 lock\_release()를 이용하여 동기화 해준다. **- pintos/src/userprog/process.c  
   :** parent, child thread를 생성하고 종료할 때 정상적으로 진행되도록 (zombie, orphan process가 발생하지 않도록) load, exit, memory에 관한 semaphore를 만들어서 process\_exit, process\_wait에서 exit, memory semaphore를 사용하고 start\_process, process\_execute 함수에서  load semaphore를 사용하여 적절히 싱크를 맞춰줄 것이다.  
   1. **연구원 역할 분담**

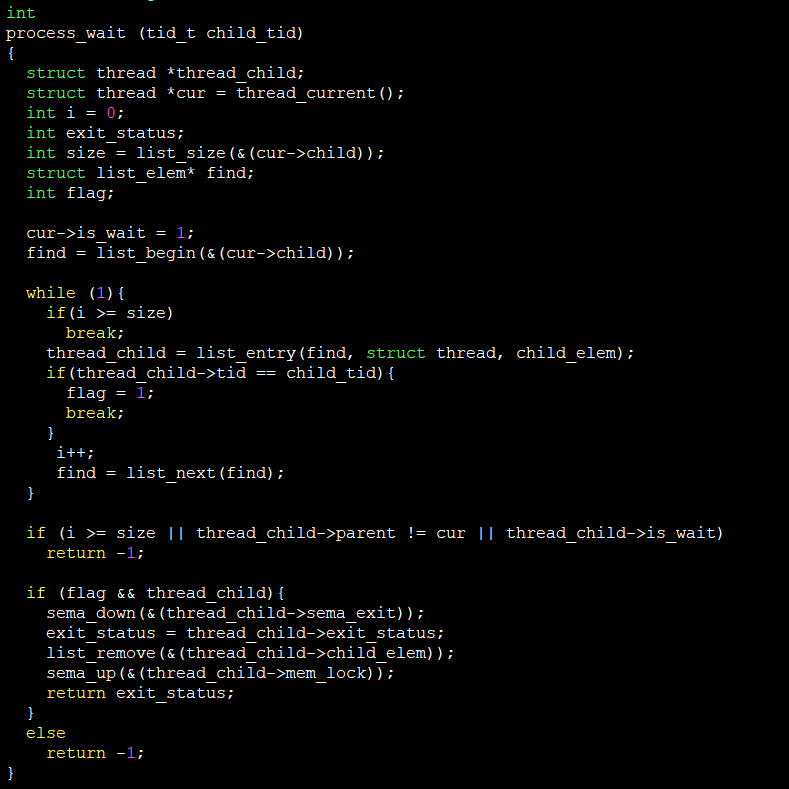
|  |  |
| --- | --- |
| 박수진 | 하민주 |
| * 매뉴얼 및 자료 분석 * 자료구조 및 알고리즘 구상 * filesystem function 구현 * synchronization 구현 * denying write to executable files 구현 * 테스트 및 디버깅 * 보고서 작성 | * 매뉴얼 및 자료 분석 * 자료구조 및 알고리즘 구상 * filesystem function 구현 * synchronization 구현 * denying write to executable files 구현 * 테스트 및 디버깅 * 보고서 작성 |

1. **연구 결과**
   1. **합성 내용**

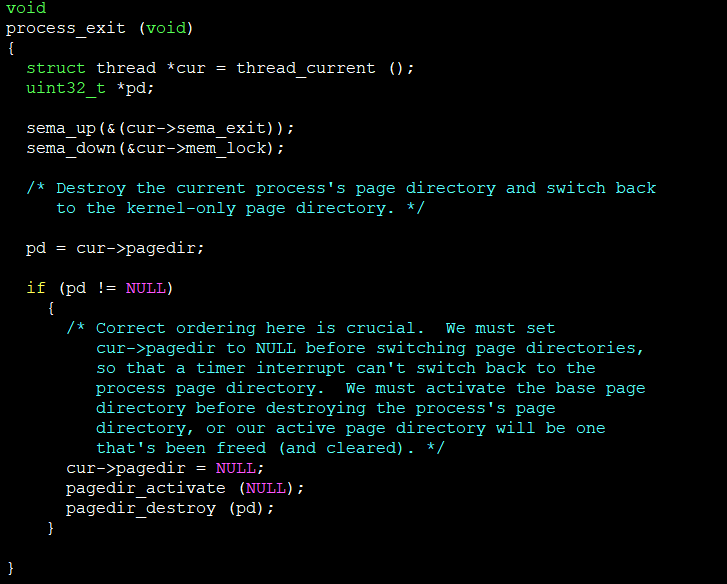
* ****
  1. **제작 내용**
     1. Semaphore
* thread 구조 수정 및 semaphore 초기화  
  **  
  **

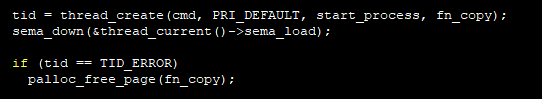
기존에 있던 semaphore외에 wait 과정에서 child의 메모리를 보호할 struct semaphore mem\_lock을 추가하였다. 그리고 sema\_init함수를 통하여 0으로 초기화 해주었다.

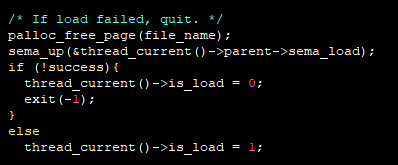
* process\_wait

  
  
1차 프로젝트에서의 semaphore를 보완하였다. semaphore 두개로는 syn-read syn-write의 구현이 어려울 것이라 판단하여 child exit\_status를 semaphore sema\_up(mem\_lock)을 걸어 부모가 list\_remove를 수행시킬 때 까지 메모리를 보호할 수 있게 하였다.

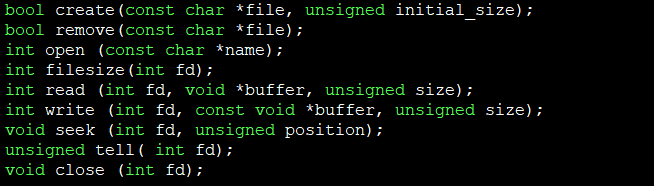
* process\_exit

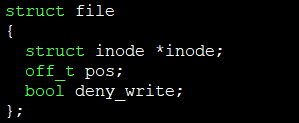
  
자식thread의 수행이 끝나고 나면 sema\_up(sema\_exit)을 통해 풀어주었다. 자식 thread가 바로 메모리 해제되면 부모가 종료상태인 줄 모를 수 있기 때문에 sema\_down(mem\_lock)을 통해 부모에게 알렸다.

- process\_execute  
  
child thread가 생성된 후 start\_process가 수행되는데 그 전에 parent thread의 process\_execute가 끝나면 parent가 child의 load 성공 유무를 알 수 없다. 즉, parent thread는 child thread의 load가 끝날 때까지 기다려야한다. 이를 구현하기 위해 process\_execute에서 child thread를 생성한 후 child가 load될 때 까지 parent를 sema\_down 시켜준다.

- start\_process  
  
child thread의 load가 완료되면 parent thread가 진행될 수 있도록 start\_process에서 parent thread를 sema\_up 시켜줘야한다.

* + 1. file system

  
▲ syscall.c에서 구현한 file system 함수

  
▲ syscall.c에서 file 구조체 추가

- thread 구조 수정  
thread.h에서 thread에  struct file file[130]을 추가하고, thread.c의 init\_thread()에서 반복문을 통하여 배열을 NULL로 초기화 하였다.

- create  
파일을 생성하는 system call function이다. 파일 명과 사이즈를 인자로 받고 만약 파일 명이  NULL이면 exit(-1)을 수행하였다. 그렇지 않으면 filesys\_create()를 호출 하였다.

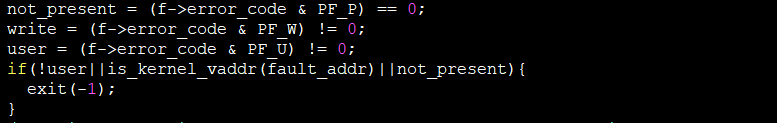
- remove  
파일을 제거하는 system call function이다. 제거할 파일명을 인자로 받아서 filesys\_remove()함수에 인자로 넘겨 호출한다.

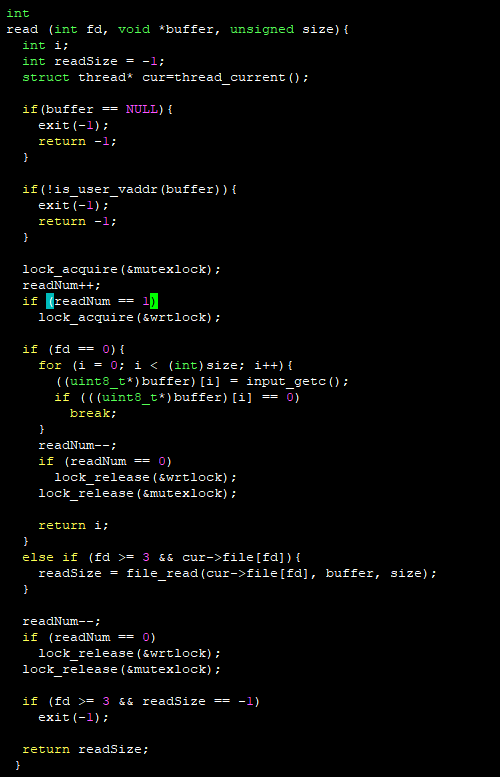
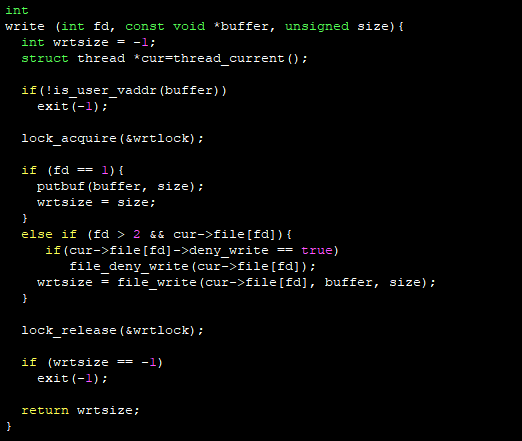
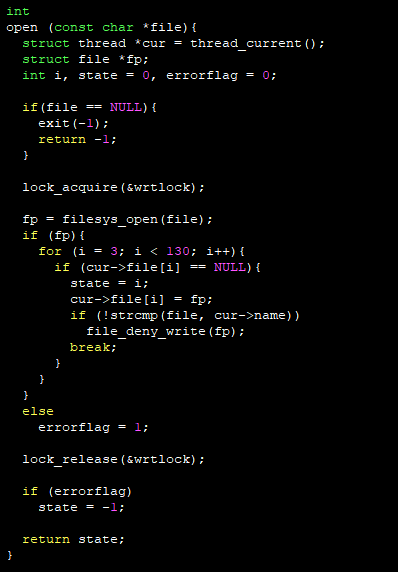
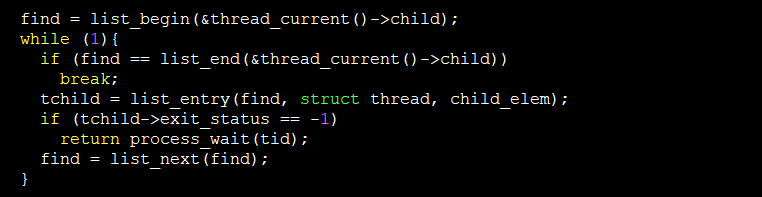
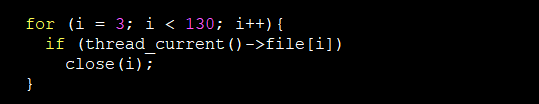
- open  
파일을 여는 system call function이다. 파일명의 유효성을 검사한후, reader/writer 문제를 해결하기 위하여 lock을 걸어준다. 그리고 인자로 받은 파일명을 가진 파일의 유효성을 확인한다. 그 후 file descriptor 배열에서 비어있는 곳에 open 한 file의 포인터를 넘겨준다. 만약 이미 실행중인 파일을 열려고하면 file\_deny\_write() 함수를 통해 권한을 제한한다. 리턴할때는 락을 해제하고 리턴해주어 다른 thread가 critical section에 진입할 수 있게 해준다.

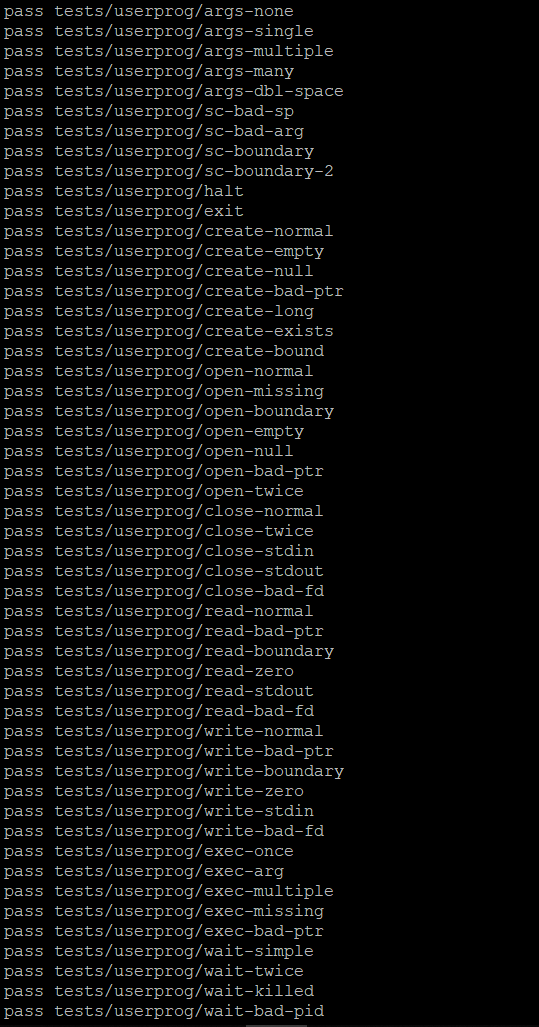
- filesize  
파일의 크기를 반환하는 system call function이다. file descriptor를 받아서 file\_length()에 인자로 넘겨 호출하면 filesize가 반환된다.

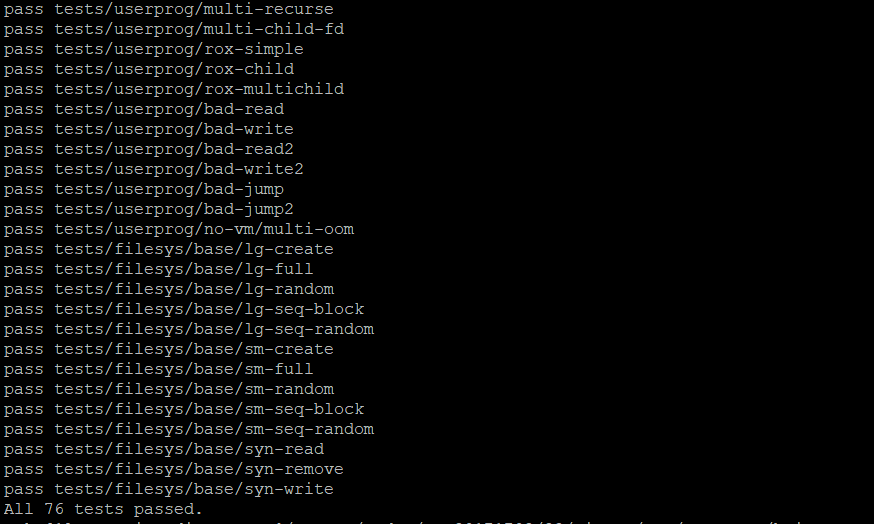
- seek  
현재 파일 포인터의 위치와, 원하는 위치(position)를 인자로 받고, file\_seek함수에 적절히 넘겨주면 원하는 위치를 찾아갈 수 있다.  
  
- tell  
현재 파일포인터의 위치를 반환하는 system call function이다. file descriptor를 인자로 받아 file\_tell()에 넘겨주어 리턴한다.  
  
- close  
파일을 닫는 system call function이다. file descriptor를 가진 file 포인터를 close해준다.  file descriptor가 NULL이면 예외처리를 해준다. 그리고 struct file \*fp라는 임시변수에 현재 닫고자 하는 file의 file descriptor를 저장하고, file descriptor를 NULL로 초기화해준다. 그 후 닫고자 하는 file descriptor를 저장한 fp를 file\_close에 인자로 넘겨주어 file을 close한다.

* + 1. NULL address problem (bad-read/bad-write)

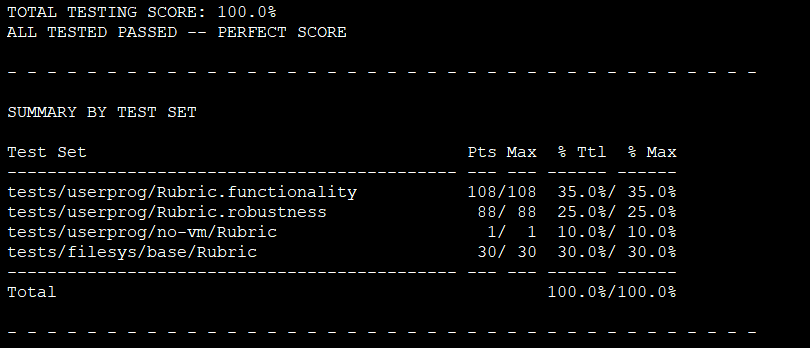
  
stdin/stdout으로 NULL이 기본 주소로 들어오면 page 접근 에러가 발생할 수 있다. not\_present인 경우 exit(-1)이 되도록 exception.c의 page fault 함수 안에서 조건을 강화하였다.

* + 1. Synchronization problem (syn-read/syn-write)  
         
         
         
         
       파일에 하나의 process만이 접근할 수 있어야 synchronization 문제가 발생하지 않는다. 이를 해결하기 위해 read, write, open할 때 lock을 걸어주어 다른 process가 접근하지 못하게 하고 function이 종료될 때 lock을 풀어주었다.
    2. Memory leak problem (multi-oom)  
       thread가 계속 fork 되고 exit 되다보면 thread의 메모리가 제대로 회수되지 않아 메모리 누수 현상이 발생할 수 있다. 메모리 누수가 발생하는 운영체제는 좋은 운영체제라고할 수 없다. 이를 방지하기 위해서는 1) semaphore로 process의 load와 exit을 적절히 조절하고 2) 할당한 메모리들은 반드시 제때 해제해주어야한다.   
         
       1) 만약 parent process가 child를 생성하고 child가 load되기 전 parent process가 종료된다면, child가 orphan process가 되어 메모리가 회수되지 않는 문제가 발생할 것이다. 이런 현상을 예방하기 위해 load에 대한 semaphore를 이용하여 thread create를 수행하고 child가 load될 때까지 기다리도록 구현하였다. 또한 orphan process를 방지하기 위해 parent가 exit되기 전에 exit\_status가 -1인 child를 wait()을 호출하여 정상적으로 회수하였다.  
         
       ▲ process\_execute 수정(강제 종료된 child list가 있는지 검사, 실패한 프로세스를 회수)  
         
       2) malloc한 모든 메모리들은 반드시 free로 다시 해제해주어야 메모리 누수가 발생하지 않는다. 또한 이번 프로젝트에서 thread에 file descriptor가 추가되었기 때문에 process가 종료되기 전에 syscall.c의 exit함수에서 반복문을 통하여  file descriptor들을 모두 close 해주어야한다.  
         
       ▲ syscall.c에서 exit수정(file descriptor close)
  1. **시험 및 평가 내용  
     -make check 결과**





**-make grade 결과**



1. **기타**
   1. **연구 조원 기여도**

* 박수진(50%), 하민주(50%)
  1. **소감**
* **박수진:** 1번 프로젝트를 할 때는 pintos가 생소해서 어디를 어떻게 해야 할지 몰라서 힘들었는데 2번 프로젝트를 할 때는 프로젝트의 요구사항이 뭔지 저번보다 이해하기 쉬웠다. 1번 프로젝트 때 wait을 처리하기 위해 semaphore를 사용했었는데 이번에 make check를 해보니 synchronization이 완벽하게 구현돼있지 않아서 다시 수정하느라 애먹었다. 여러 process를 돌리는 것은 정말 힘든 일인 것 같다. 그리고 수업시간에 배운 readers-writers problem을 이번에 적용했더니 syn-read와 syn-write가 바로 통과돼서 기분이 좋았다. test case 중에 multi-oom 때문에 너무 힘들었는데 한참을 수정하다가 perfect score가 뜨는거 보고 행복해서 눈물이 날 것 같았다.
* **하민주:** 이번 프로젝트는 지난 플젝인 user program 1 을 통해 개념을 잡고 시작해서 그런지 접근하는 것이 어렵지는 않았다. 그래서 file read,write,open,close 등은 비교적 쉽게 구현할 수 있었다. 하지만 synchronization이 process 내에서  sema를 이용하여 싱크가 잘맞게 수정해야하다 보니까 무척 어려웠다. 한줄만 잘못추가해도 프로그램이 안돌아가서 너무 스트레스를 받았다. 다행히 무사히 프로젝트를 마칠 수 있었고,다음 플젝이 제일 어렵다는데 너무 무섭다. 쉬운 축에 속하는 이번 프로젝트도 겨우 해냈는데, 다음 플젝은 좀 더 일찍 시작해야겠다는 생각을 하였다.