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

**(설계 프로젝트 수행 결과)**

과목 명 : 운영체제

담당 교수 : 김영재 교수님

조 / 조원 : 72조 20141515 김연후 20141362 천지환

개발 기간 : 2019.11.02 ~ 11.16

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

**제출일 : 2019.11.17**

**참여 조원 : 20141515 김연후, 20141362 천지환**

1. **개발 목표**

* 저번 project1에 이어서 syscall.c에 있는 system call handler을 마저 완성시켜준다. 이번에 완성시켜야 했던 system call은 file system과 관련있는 system call로써 create, remove, open, close, filesize, seek, tell 이 있었으며 저번에 구현하였던 read 와 write가 file system과 관련하여서도 작동할 수 있도록 수정해준다. 이때 synchronaization과 read only로 file을 open 하는 것도 고려하여 준다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* 저번에 syscall에 작성하였던 standard input output에 관한 함수 및 기본적인 os syscall 구현에 이어서 이번 project2에서는 file system과 관련된 함수들을 구현하여야 했기에 코드 구현은 주로 syscall.c에서 이루어졌다.

1. 기본적인 file system과 관련된 system call 구현

-src/userprog/syscall.c

저번 project1에서 standard input output과 기본적인 os 구현에 필요한 sytem call들을 완성시켜주었다 이에 이어서 project 2에서는 file system에 관한 function들(create, remove, open close, filesize, seek, tell)을 구현해 준다.

-System call 분류 및 처리

1. SYS\_CREATE : 입력받은 파일 이름과 크기에 대한 파일을 생성시켜준다. 생성 성공 시 TRUE를 실패시 FALSE를 반환시켜준다.
2. SYS\_REMOVE : 입력받은 파일을 제거 하고 성공시 TRUE를 실패시 FALSE를 반환시켜준다.
3. SYS\_OPEN : 입력받은 파일을 생성하고 파일 생성에 성공하였을시 생성한 파일의 file descriptor(3~128의 int값)을 반환하고 실패시 -1을 반환한다.
4. SYS\_CLOSE : 입력받은 file descriptor을 이용해 file을 종료시킨다..
5. SYS\_READ : standard input을 확장해 file에 대한 read도 가능하도록 해준다.
6. SYS\_WRITE: standard output을 확장해 file에 대한 write도 가능하도록 해준다
7. SYS\_FILESIZE: 입력받은 file descriptor을 통해 file의 사이즈를 반환시켜준다.
8. SYS\_SEEK: 입력받은 file descriptor와 position을 통해 다음 읽거나 쓸 file의 위치를 position에 해당하는 위치로 바꿔준다.
9. SYS\_TELL: 입력받은 file descriptor에 대한 file의 다음 읽을 위치나 다음 쓸 위치를 반환시켜준다.
10. Read, write 수정

-src/userprog/syscall.c

지난 project1에서 우리는 기본적인 read와 write을 구현해 주었다. 하지만 이는 file descriptor가 0,1일 때 즉 standard input과 output일 때에 대한 구현이였기에 이를 확장하여 기본 file들에 대한 read write을 시행할 수 있게 구현한다.

1. Rox 구현

-src/userprog/syscall.c

현재 실행중인 파일을 다른 file이 write할 수 없게한다. 이는 child에 해당하는 process들에 대해서도 마찬가지이다.

1. Synchronize 구현

-src/userprog/syscall.c, src/userprog/process.c

Critical section에 대해서 현재 우리가 그 critical section에 해당하는 작업을 하고 있을 경우 다른 process들이 이를 사용할 수 없도록 critical section에 대한 작업을 해준다.

1. Multi-oom 구현

-src/userprog/process.c, src/userprog/thread.h , src/userprog/thread.c

Child를 한계까지 만든다음에 그 child를 다 닫는 작업이 정상적으로 처리되도록 해준다.

* 1. **개발 내용**

-기본적인 file system과 관련된 system call 구현

-src/userprog/syscall.c

Syscall에 있는 syscall\_handler에 file관련된 함수를 이용해 file system을 완성한다.

-System call 분류 및 처리

1. SYS\_CREATE : 구현 내용은 개발 범위에서 설명한 바와 같고 files에 구현되어 있는 함수인 filesys\_create 함수를 이용해준다.
2. SYS\_REMOVE : 구현 내용은 개발 범위에서 설명한 바와 같고 filesys에 구현되어 있는 함수인 filesys\_remove 함수를 이용해준다.
3. SYS\_OPEN : 구현 내용은 개발 범위에서 설명한 바와 같고 filesys에 구현되어 있는 함수인 filesys\_open 함수를 이용하고 비어있는 file descriptor(3~127 중 하나)에 file pointer를 넣어준다.
4. SYS\_CLOSE : 구현 내용은 개발 범위에서 설명한 바와 같고 file에 구현되어 있는 함수인 file\_close 함수를 이용해준다. 그 후 해당 dile descriptor를 NULL로 초기화해준다.
5. SYS\_FILESIZE: 구현 내용은 개발 범위에서 설명한 바와 같고 file에 구현되어 있는 함수인 file\_length 함수를 이용해준다.
6. SYS\_SEEK: 구현 내용은 개발 범위에서 설명한 바와 같고 file에 구현되어 있는 함수인 file\_seek 함수를 이용해준다.
7. SYS\_TELL: 구현 내용은 개발 범위에서 설명한 바와 같고 file에 구현되어 있는 함수인 file\_tell 함수를 이용해준다.

-Read, write 수정

-src/userprog/syscall.c

1. SYS\_READ : 구현 내용은 개발 범위에서 설명한 바와 같고 입력받은 file descriptor가 3~127사이의 값을 가지고 있을 경우 file\_read를 통해 해당 file의 내용을 buffer에 읽어드리고 읽은 횟수를 반환한다. 실패할 시 -1을 반환한다.
2. SYS\_WRITE: 구현 내용은 개발 범위에서 설명한 바와 같고 입력받은 file descriptor가 3~127사이의 값을 가지고 있을 경우 file\_write를 통해 해당 buf의 내용을 file에 쓰고 쓴 횟수를 반환한다.실패할 시 -1을 반환한다.

-Rox 구현

-src/userprog/syscall.c

구현 내용은 개발 범위에서 설명한 바와 같고 open 함수에서 현재 열린 파일과 실행중인 파일이 같을시 file\_deny\_write 함수를 이용해 file structure의 deny\_write flag를 set해주어서 write을 막아주고 child의 접근도 막아준다.

- Synchronize 구현

-src/userprog/syscall.c, src/userprog/process.c

구현 내용은 개발 범위에서 설명한 바와 같고 syschronization apis 중 하나인 lock을 이용하여 open, read, write에 lock을 걸어 critical section에 동시에 접근하지 못하도록한다. 이때 process에서 file을 open하는 execute와 load에도 loack을 걸어주어야 한다.

-Multi-oom 구현

-src/userprog/process.c, src/userprog/thread.h , src/userprog/thread.c

구현 내용은 개발 범위에서 설명한 바와 같고 thread가 load 되기 전에 먼저 닫혀서 에러나는 경우를 방지하기위해 load와 관련된 semaphore 변수를 thread에 추가하여 이를 synchronize해주고 load에 실패할 시 exit(-1)을 해준다. 그 후 강제 종료된 child list가 있다면 이를 회수해준다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

|  |  |
| --- | --- |
| 기간 | 구현 내용 |
| 2019.11.3.~6 | Pintos ppt 및 Pintos Manual 이해 및  기본적인 create, remove, open close, filesize, read, write, seek, tell 구현 |
| 2019.11.7~11 | os시험공부 다른 프로젝트로 인한 지연 |
| 2019.11.12. | 작성한 system call들의 뼈대에서 test들을 하면서 에러 체크 및 수정 |
| 2019.11.13 | Bad read, write, jump들의 에러 체크 및 코드 수정 |
| 2019.11.14 | Rox 관련 test들의 에러 체크 및 코드 수정 |
| 2019.11.15 | Multi-oom test 에러 체크 및 코드 수정 |
| 2019.11.16 | Pintos 보고서 작성 |

* 1. **개발 방법**

- 기본적인 내용은 II의 개발 범위와 내용에 기술되어 있으며 여기서는 거기에 추가하여 구현부분을 조금 더 자세히 다룬다. 실제 구현 CODE는 IV에서 다루었다.

-기본적인 file system과 관련된 system call 구현

|  |
| --- |
| 사용한 file 관련 함수 |
|  |

-src/userprog/syscall.c

Syscall에 있는 syscall\_handler에 file관련된 함수를 이용해 file system을 완성한다.

-System call 분류 및 처리

1. SYS\_CREATE : 먼저 bad access를 방지하기 위해 입력받은 file pointer가 user process의 memorry영역이 아닐경우 exit(-1)을 해주고 file pointer가 NULL일 경우에서 exit(-1)을 해준다. 그 후 입력받은 file pointer 와 initial size를 parameter로 filesys\_create을 시행해주며 성공할시 true를, 실패할시 false를 return해준다.
2. SYS\_REMOVE : : 먼저 bad access를 방지하기 위해 입력받은 file pointer가 user process의 memorry영역이 아닐경우 exit(-1)을 해주고 file pointer가 NULL일 경우에서 exit(-1)을 해준다. 그 후 입력받은 file pointer을 parameter로 해서 filesys\_remove를 해주고 이를 성공할 시 true를 실패할시 false를 return해준다.
3. SYS\_OPEN : : 먼저 bad access를 방지하기 위해 file pointer가 NULL일 경우에서 exit(-1)을 해준다. 그 후 filesys\_open함수에 입력받은 file pointer을 parameter로 넣어주어 file을 열어 주며 이때 file이 NULL일 경우 -1을 return 하고 아닐 경우 thread\_current()를 이용해 현재 돌아가고 있는 thread의 fd에 빈자리를 찾아 file을 넣어주고 이 file descriptor을 반환해준다. 이것 외의 경우에는 -1을 return해준다.
4. SYS\_CLOSE :입력 받은 file discriptor에 해당하는 file을 현재 돌아가고있는 thread에 대해 thread\_current()를 이용해 찾는다. 이때 찾은 file이 NULL일 경우에 exit(-1)을 해주고 아닐 경우 찾은 file pointer을 parameter로 해 file\_close 함수를 시행해준 다음 해당 file descriptor을 NULL로 바꿔준다.
5. SYS\_FILESIZE: 입력 받은 file discriptor에 해당하는 file을 현재 돌아가고있는 thread에 대해 thread\_current()를 이용해 찾는다. 이때 찾은 file이 NULL일 경우에 exit(-1)을 해주고 아닐 경우 찾은 file pointer을 이용 해 file\_length 함수를 시행해주고 이 값을 return한다.
6. SYS\_SEEK: 입력 받은 file discriptor에 해당하는 file을 현재 돌아가고있는 thread에 대해 thread\_current()를 이용해 찾는다. 이때 찾은 file이 NULL일 경우에 exit(-1)을 해주고 아닐 경우 찾은 file pointer와 position을 parameter로 해 file\_seek 함수를 시행해준다.
7. SYS\_TELL: 입력 받은 file discriptor에 해당하는 file을 현재 돌아가고있는 thread에 대해 thread\_current()를 이용해 찾는다. 이때 찾은 file이 NULL일 경우에 exit(-1)을 해주고 아닐 경우 찾은 file pointer을 parameter로 해 file\_tell 함수를 시행해서 position을 찾은 다음 이를 return 해준다.

-Read, write 수정

-src/userprog/syscall.c

1. SYS\_READ : 입력받은 fd가 0일 경우 저번에 구현해주었던 read를 시행하고 fd가 3~127사이의 값을 갖을 경우 입력받은 fd를 이용해 현재 실행중이 thread의 thread\_current()->fd[fd]에서 file을 연다 이때 file pointer가 NULL일 경우 exit(-1)을 반환 해주고 아닐 경우 file\_read에 file, buf, size를 parameter로 시행해준다. 이 떄 시행해준 결과로 read해준 숫자를 반환해주는데 이를 return해준다.
2. SYS\_WRITE: 입력받은 fd가 1일 경우 저번에 구현해주었던 read를 시행하고 fd가 3~127사이의 값을 갖을 경우 입력받은 fd를 이용해 현재 실행중이 thread의 thread\_current()->fd[fd]에서 file을 연다 이때 file pointer가 NULL일 경우 exit(-1)을 반환 해주고 아닐 경우 file\_wite에 file, buf, size를 parameter로 시행해준다. 이 떄 시행해준 결과로 write해준 숫자를 반환해주는데 이를 return해준다.

-Rox 구현

-src/userprog/syscall.c

|  |
| --- |
| 사용한 file deny 함수 |
|  |

Open 함수에서 file을 open할 때 현재 돌아가고 있는 thread의 name과 입력받은 file name을 비교하여 이들이 같으면 file\_deny\_write 함수를 filesys\_open을 통해 open한 file을 parameter로 넣어서 시행해준다. 이를 통해 write이나 child의 접근을 막아준다. 이때 thread의 name은 thread\_current()->name을 이용해서 얻는데 name이 parsing이 안되어 있을 수 있기에 저번 project1에서 하였던 것처럼 parsing을 해준다.

- Synchronize 구현

-src/userprog/syscall.c, src/userprog/process.c

|  |
| --- |
| Synchronize 문제가 발생하는 critical section에 대한 사진 |
|  |
| Synchronize에 사용한 lock함수 |
|  |

먼저 critical section을 보호하기 위해 lock structure에 대한 변수를 전역변수로 설정해준다. 그 후 File 을 open, read write 할때 마다 lock\_acquire과 lock\_release 함수를 이용해 (아까 설정한 전역변수 lock을 parameter로 넣어서) critical section을 보호해준다. Open, read, write 함수의 코드 상 각각 file이 NULL일때의 예외처리를 해준 뒤 lock acquire을 걸어준뒤 return하기전 lock release를 해주어 그때부터 다른 process나 file도 critical section을 사용할 수 있도록 해준다. 이때 process.c에 있는 process\_execute나 load에 있는 함수의 file\_open 전후에도 lock\_acquire과 lock\_release를 이용해서 보호해준다.

-Multi-oom 구현

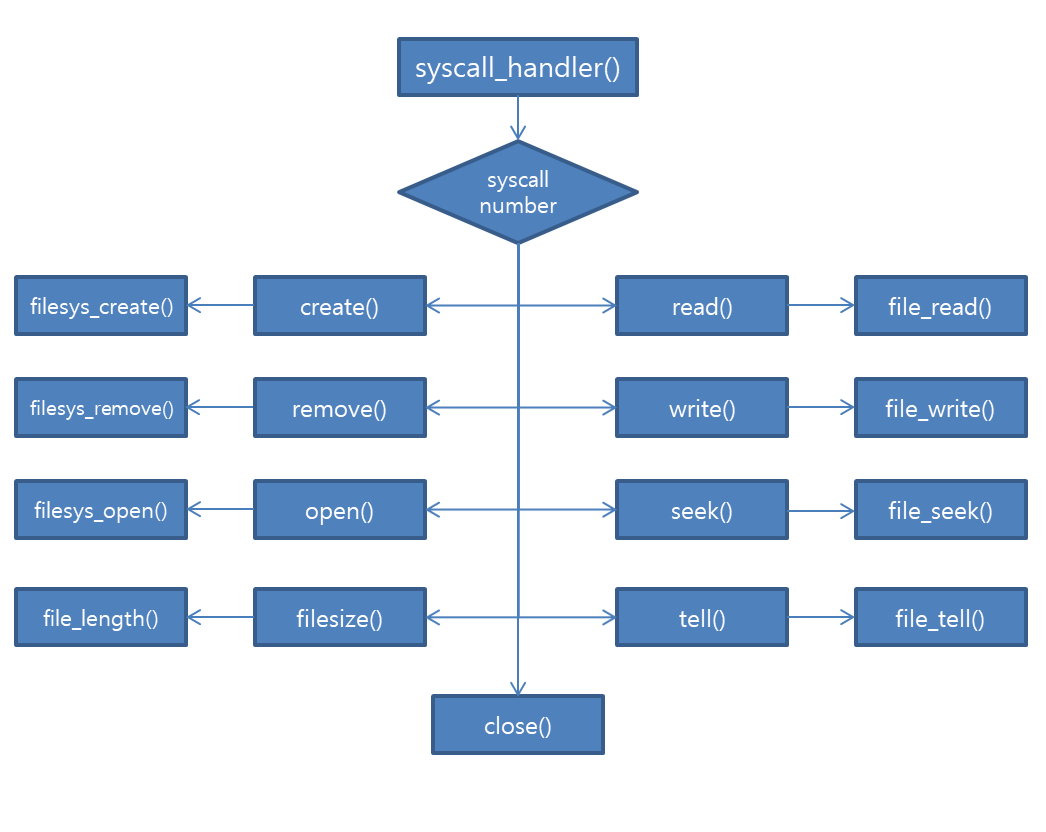
-src/userprog/process.c, src/userprog/thread.h , src/userprog/thread.c

Multi-omm test를 시행시켜보면 처음에 “crashed child should return -1”이 나온다. 이를 pintos/src/tests/userprog/no-vm에 가서 찾아보면 crash된 child에 대해 wait(child\_pid)가 -1을 return 하지 못하였기에 생성된 오류임을 알 수 있다. 따라서 wait관련 문제임을 알 수 있는 데 이는 thread가 만들어 지고 너무 빨리 종료되어 load 되기도 전에 종료되는 문제가 발생할 수 도 있음을 유추할 수 있고 따라서 Thread에 lock과 관련된 semaphore 변수를 추가하여 이를 process.c에 있는 process\_excute의 thread\_create 부분에서 sema\_down을 시행해 thread가 load되고 있을 시 parent가 기다리도록 해주었다. 그 후 sema\_up start\_process의 load이후에 시행하여 lock을 풀어 주었고 load에 실패할시 exit(-1) 명령어를 시행해주었다. 그 뒤 강제 종료된 child들이 thread의 child list 목록에 남아 있을 수 있기에 child list 목록을 돌면서 exit\_status 가 -1일 child 들을 회수해주었다.

* 1. **연구원 역할 분담**
* 각자 project 2를 구현한다음 만나서 더 코드 구성이 좋은 프로그램 제출, 보고서는 나눠서 작성

1. **연구 결과**
   1. **합성 내용**

Making System Calls (Additional System Calls)



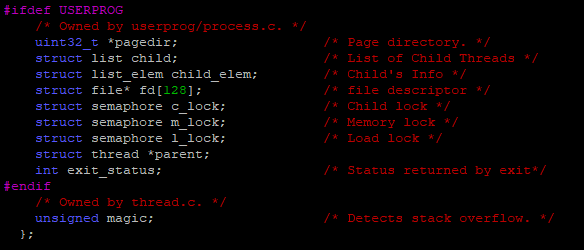
user program level에서 file과 관련된 함수들을 호출한다.

File 관련의 함수 호출 시 system call을 통해 사용자가 요청한 작업을 수행한다. 예를 들어 open, close를 보자. open의 경우 먼저 user program에서 fopen 함수가 호출되면 내부적으로 system call인 open()함수를 호출한다. 그러면 system call handler에 의하여 직접 만든 open() 함수가 수행되며, 그 내부의 filesys\_open() 함수를 호출하여 file open 작업을 수행하게 된다. 이때 file을 관리하기 위해 필요한 정보를 thread\_current()에 저장한다. file을 이용한 작업이 모두 완료되면, close()함수를 호출하여 file\_close() 함수가 호출되고, file descriptor 등의 여러 변수들을 사용 해제하여 즉, process가 끝나기 전에 열린 file에 대해 file\_close() 함수를 이용하여 닫아주게 된다.

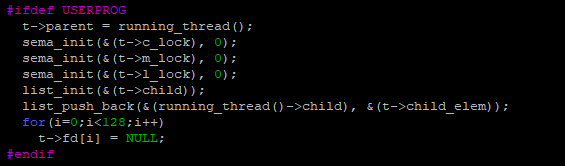
* 1. **제작 내용**

thread에 file 관련 정보를 관리하기 위하여 file descriptor 정보와 실행중인 file 정보를 저장하도록 thread 구조체를 조정하였다. 또한 wait을 위한 semaphore들도 구현하였다.

thread.h



thread.c



System Call의 create, remove, open, close, read, write, filesize, seek, tell을 구현하였다.

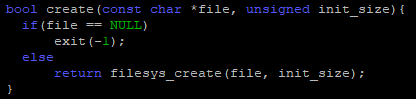
syscall.c, syscall.h

* create (SYS\_CREATE)

파일을 생성하는 시스템 콜이며, 파일 생성 성공 시, true, 실패할 경우 false를 return한다.

file : 생성할 파일의 이름 및 경로 정보

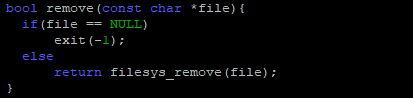
init\_size : 생성할 파일의 크기



* remove (SYS\_REMOVE)

파일을 삭제하는 시스템 콜이며, 파일 제거 성공 시 true, 실패할 경우 false를 return한다.

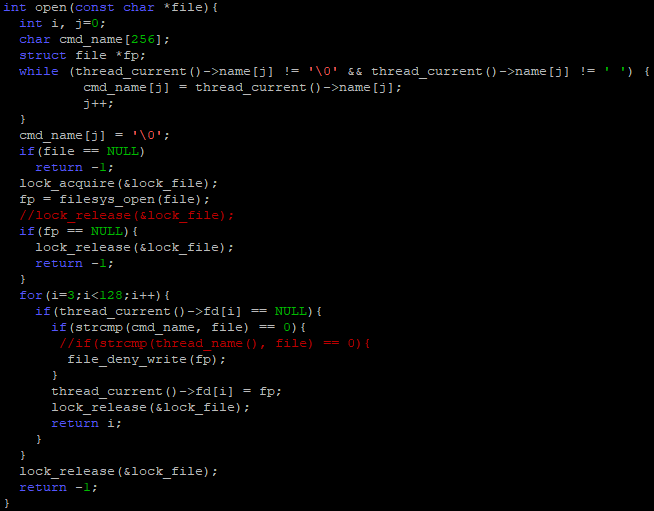
file : 제거할 파일의 이름



* open (SYS\_OPEN)

파일을 여는 시스템 콜이며, 현재 thread를 찾아서 thread의 file descriptor table에 현재의 file을 연결해준 뒤 연결한 file descriptor의 index를 return한다.

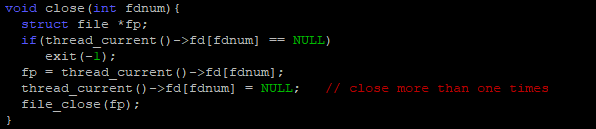
file : 현재 열려고 하는 파일



* close (SYS\_CLOSE)

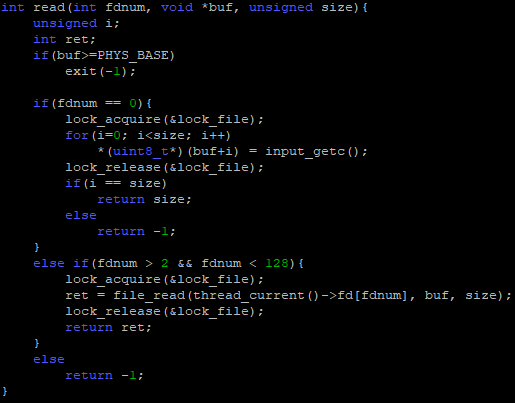
파일을 닫는 시스템 콜이며, 넘겨받은 file descriptor index를 갖는 file을 close해주는 함수다. 현재 thread의 file descriptor table에서 fdnum을 이용하여 file에 접근해 file\_close()로 닫는다. 그 다음 fdnum을 index로 갖는 file descriptor table에 NULL 값을 넣어준다.

fdnum : 현재 닫으려고 하는 파일의 file descriptor index



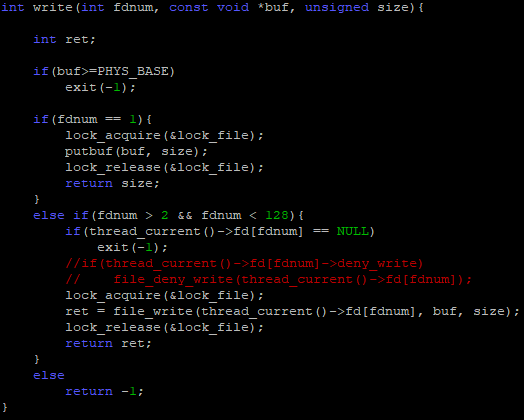
* read (SYS\_READ)

기존에 구현했던 시스템 콜 read에 lock structure를 이용하여 synchronization을 구현하였다. lock\_acquire(), lock\_release()를 이용하고, input\_getc()와 file\_read()를 사용할 때를 critical section으로 보고 lock을 걸어주었다.



* write (SYS\_WRITE)

기존에 구현했던 시스템 콜 read에 lock structure를 이용하여 synchronization을 구현하였다. lock\_acquire(), lock\_release()를 이용하고, input\_getc()와 file\_read()를 사용할 때를 critical section으로 보고 lock을 걸어주었다.



* filesize (SYS\_FILESIZE)

파일의 크기를 알려주는 시스템 콜이며 넘겨받은 file descriptor index로 현재 thread에서 file을 찾아 filesys\_filesize() 함수를 이용하여 성공 시 파일의 크기를 반환, 실패 시 -1을 반환한다.



* seek (SYS\_SEEK)

열린 파일의 위치(offset)를 이동하는 시스템 콜이며 현재 thread에서 file을 찾아서 이동시켜준다.

pos : 현재 위치(offset)를 기준으로 이동할 거리



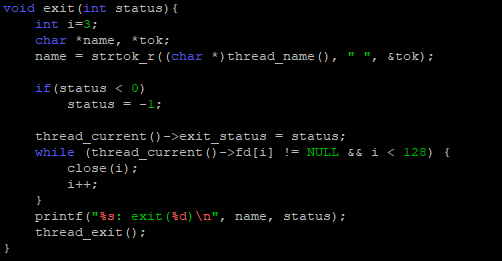
* tell (SYS\_TELL)

열린 파일의 위치(offset)를 알려주는 시스템 콜이며 현재 thread에서 file을 찾아서 위치를 알린다.



* exit (SYS\_EXIT)

프로세스가 죽을 때 지금까지 열었던 file을 자동으로 close 해주지 않기 때문에 직접 닫아주는 구현을 exit에서 해준다.



process.c 에서의 synchronization 구현

앞서 thread 구조체에서 만든 semaphore c\_lock, m\_lock, l\_lock을 이용하였다.

c\_lock의 경우 wait(child)를 호출해주는 경우에 child thread에 자신을 wait 하는 parent가 있음을 알리고, sema\_down(&c->c\_lock)으로 기다린다. 이후 child가 exit하는 경우 process\_exit에서 sema\_up(&cur->c\_lock)으로 parent가 child\_list에서 child에 대한 정보를 제거해도 된다고 알린다.

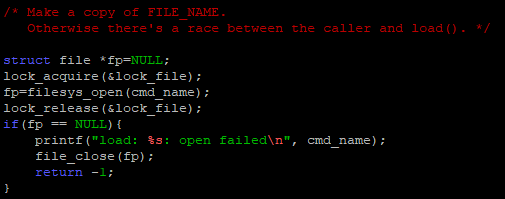
m\_lock의 경우 child는 parent가 list\_remove를 수행할 때 까지 memory는 남겨둬야 하기 때문에 process\_exit에서 sema\_down(&cur->m\_lock)을 해준 뒤, process\_wait에서 list\_remove를 한 다음에 sema\_up(&c->m\_lock)을 해준다.

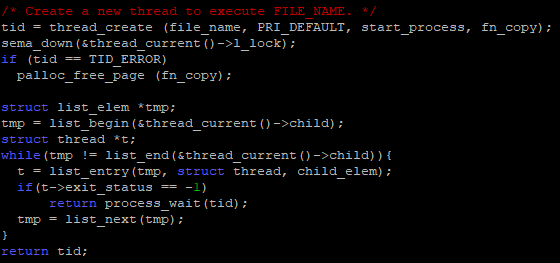
l\_lock의 경우 child thread의 정상적으로 load 됐는지 여부를 위해 사용되었다. child가 load 되기도 전에 parent가 종료되어버리면 문제가 발생하므로, load 호출 이후 sema\_up(&thread\_current()->parent->l\_lock) 으로 parent가 실행될 수 있도록 깨워준다. 이후, sema\_down((&thread\_current->l\_lock) 해준다.

이외에도 filesys\_open() 함수 사용시 앞 뒤에 lock\_acquire(), lock\_release()를 사용하여 critical section을 구현하였다.

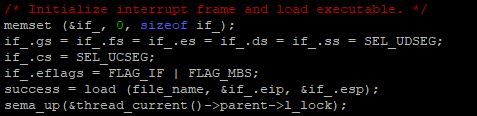
이와 같이 3가지 semaphore와 이용하여 synchronization을 구현하였다.

process\_execute 부분

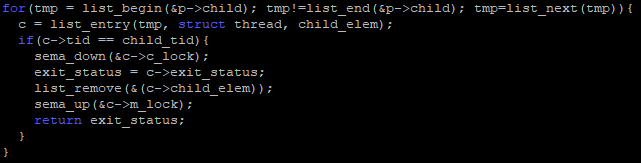




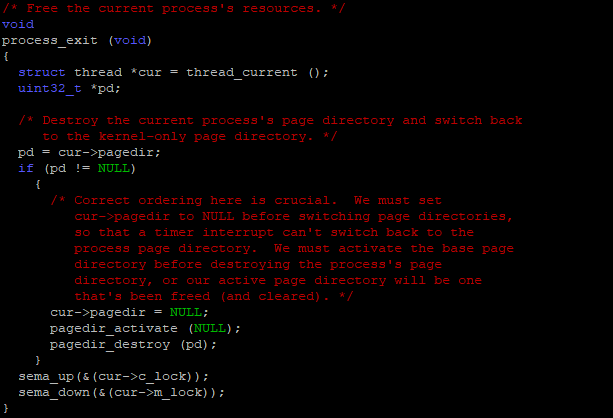
start\_process 부분



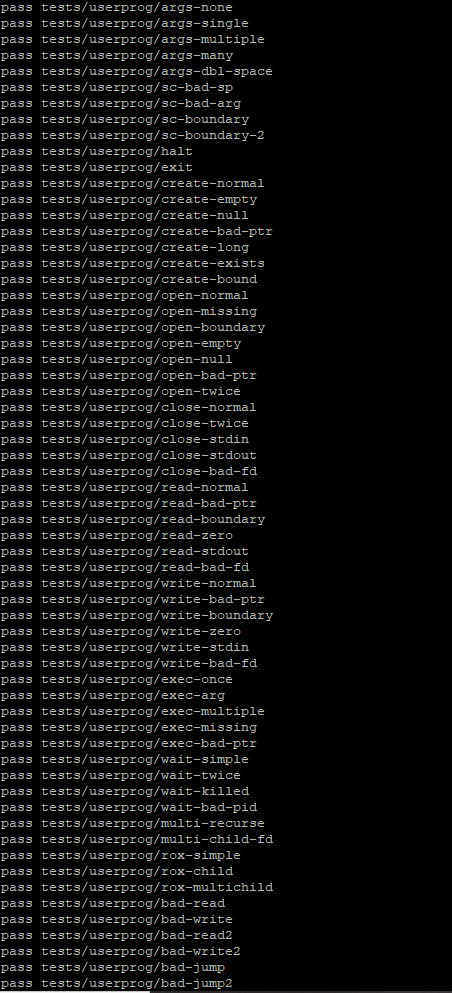
process\_wait 부분

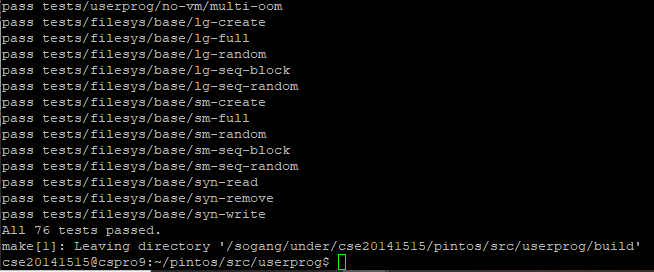


process\_exit 부분

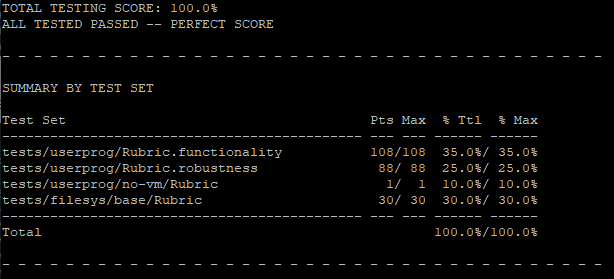


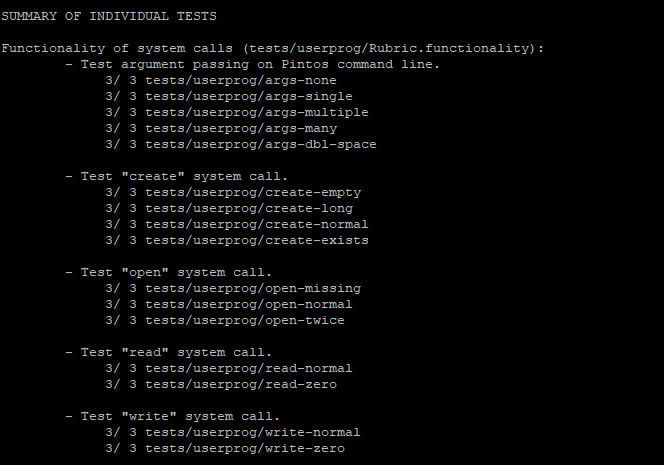
* 1. **시험 및 평가 내용**
* make check

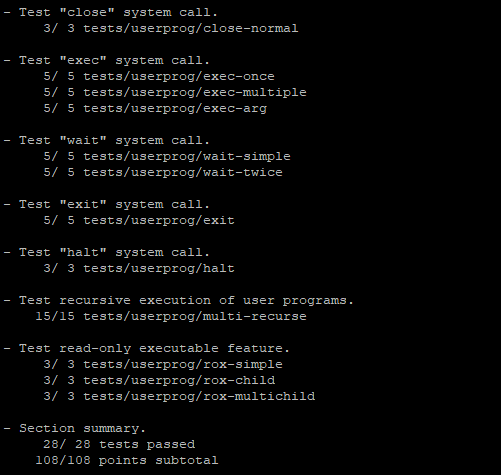


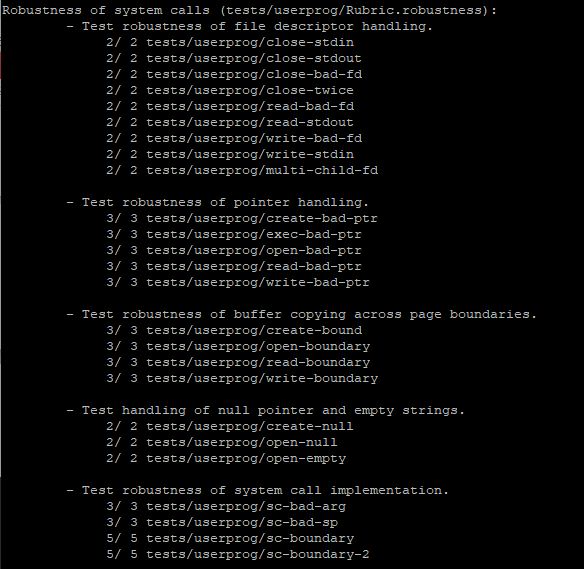


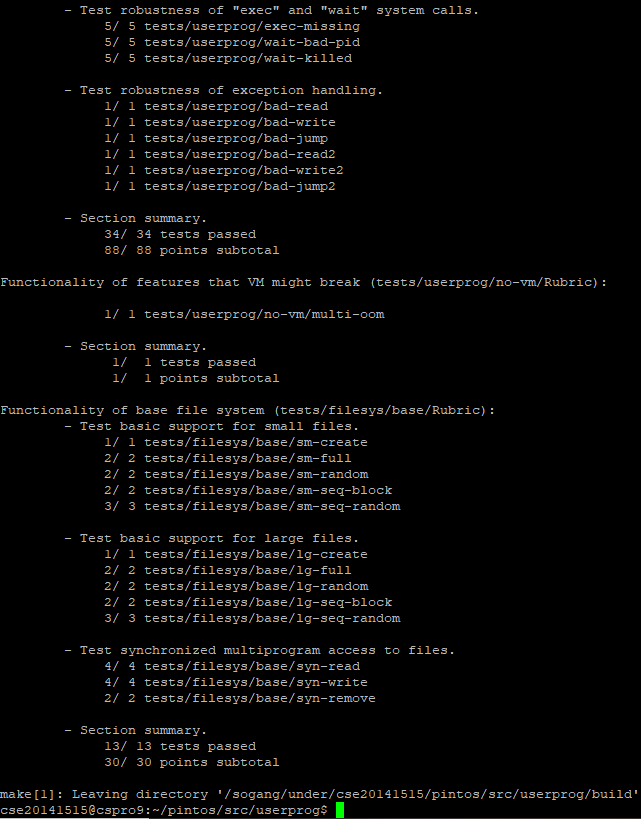
* make grade











1. **기타**
   1. **연구 조원 기여도**

* 20141362 천지환 50% 20141515 김연후 50%
  1. **소감**
* 김연후

Project 1에 비해서는 비교적으로 수월하게 system call을 구현할 수 있었던 것 같다. (이미 한번 구현해보았기에) 그러나 make check의 rox 부분과 multi-oom에서 어느 정도 고생을 하는 바람에 결국 밤새서 짜는 건 마찬가지라서 힘들었다.

* 천지환
* Project 1에서 os에 보다 익숙해진 점도 있고 고려해야하는 부분이 syscall.c의 handler 부분에 거의 집중 되어 있었기 때문에 저번보다 쉽게 코드를 구현할 수 있었다. 하지만 이번 syscall의 fail 에러를 잡아낼 때 fail이 뜨는 이유가 예상치 못한 다른 함수에서 발생하는 경우가 있어서 에러체크에 더 시간이 든 것 같다.