**Pintos Project 2 : User Program 2**

**(설계 프로젝트 수행 결과)**

**:**

과목 명 :운영체제

담당 교수 :김영재

조 / 조원 : 10조/황순, 허준형

개발 기간 :2018.10.22~2018.11.14

**프로젝트 제목 : pintos project 2 user program 2**

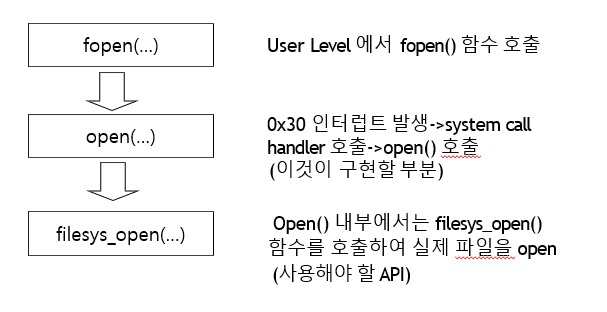
**제출일 2018.11.14 :**

**참여 조원 : 황순, 허준형**

1. **개발 목표**
2. Pintos에는 기본적인 User Program 실행환경은 구성되어 있다 지난번 프로젝트로 기본적인 시스템 콜을 구현하여 환경을 갖추었다. 이제 동기화기법을 잘 개발하여, 완벽하게 userprogram 을 구현하려고 한다. 이를 위해 핀토스의 파일 구조를 익히고 기본 API 를 숙지한다.
3. **개발 범위 및 내용**
   1. **개발 범위**

* 1. Semaphore process- waiting
* 지난번 구현은 busy-waiting을 이용하였으나, 너무 오래 기다려서 비효율적인 코드였다. 이번 프로젝트에서는 semaphore를 통해 process간 동기화를 구현하도록 한다.
* 2. 기타 파일처리 system call 구현
* -Create: file 을 생성한다.
* -Remove: 열기/닫기 와 상관없이 file을 제거한다.
* -Open: 지정한 파일을 열고, 그에 해당하는 file\_descriptor를 반환한다.
* -Close: 지정한 파일을 닫는다.
* -Filesize: 지정 파일의 사이즈를 찾는다.
* -Read: 지정 파일의 정보를 읽는다. 실제 읽은 크기의 size를 반환한다.

-Write: 버퍼에 기록되어있는 내용을 해당하는 파일에 쓴다. 그리고 크기를 반환

* -Seek: 다음에 읽을 위치 및 혹은 기록될 위치를 반환
* -Tell: 현재 파일의 기록될 위치를 반환한다.
  1. **개발 내용**
* **1. 기본적인 파일 시스템의 구조와 API 사용법을 읽고 숙지하였다.**
* -pintos manual ( pp. 31)
* 기본적인 파일시스템 구조의 흐름은 다음과 같다. (서강대 프로젝트2 참고자료)
* 

**-이를 토대로 기본적인 파일시스템 구조를 작성한다.**

**2. Denying Writes to Executables**

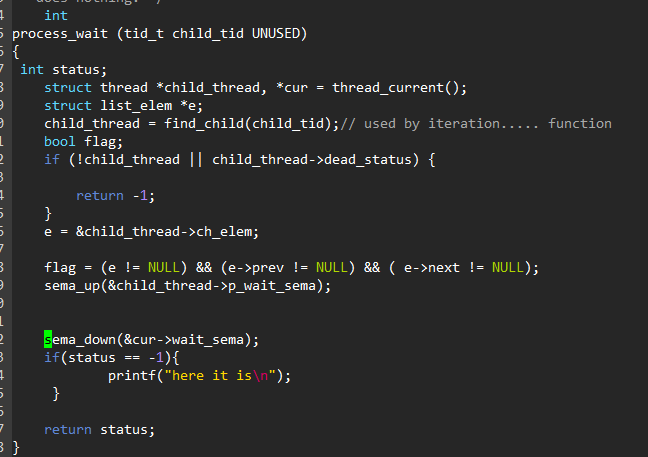
**- 프로세는 메모리에 로드되기 때문에 실행파일은 지워져도 문제없지만 핀토스는 그걸 원하지 않으므로, 적절한 api 를 이용하여 이를 구현하도록한다,**

**3. 기타 다른 동기화를 이용하여 테스트 케이스를 통과한다.**

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

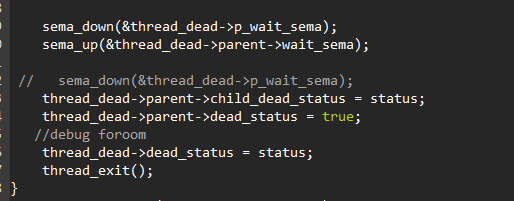
|  |  |
| --- | --- |
| **2018. 10. 22 ~ 2018.10.24** | **동기화 구현(process\_wait)** |
| **2018.10.24 ~ 2018 10.26** | **API 를 이용 파일 시스템 구조 구현** |
| **2018.10.26 ~ 2018. 11.13** | **기타 다른 테스트 케이스 통과 구현** |

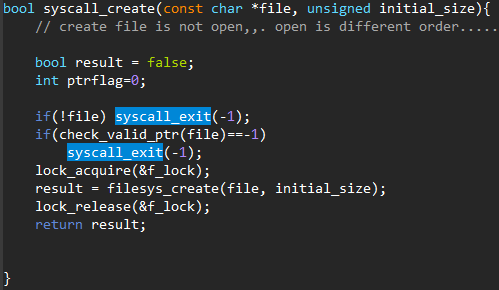
* 1. **개발 방법**
     + **프로세스 동기화 코드 구현( 세마포어)**

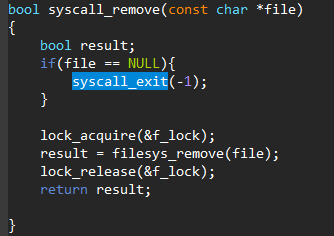


**위와 같이 프로세스 웨이트 할 때 현재 쓰레드에서 자식이 죽는걸 기다리게 한후, 다음 자식프로세스가 죽는걸 허락하는 세마포어를 구현하였다 이는 다시 systemccall\_exit에 구현되어있다.**

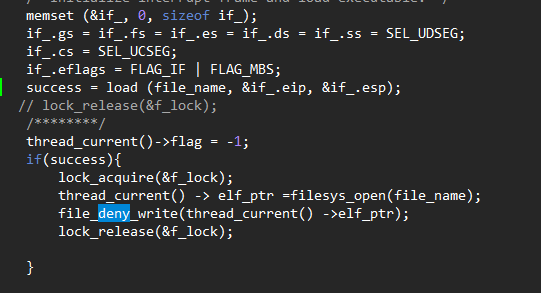
**-syscall\_Exit 에 있는 구현 부분.**



* + - **파일 시스템의 코드들 위에 말한 것들의 기본구현이다. Pintos manul을 토대로 지난번 system call 을 구현할 때 처럼 구성하였다. (주요 몇가지 코드 첨부)** 

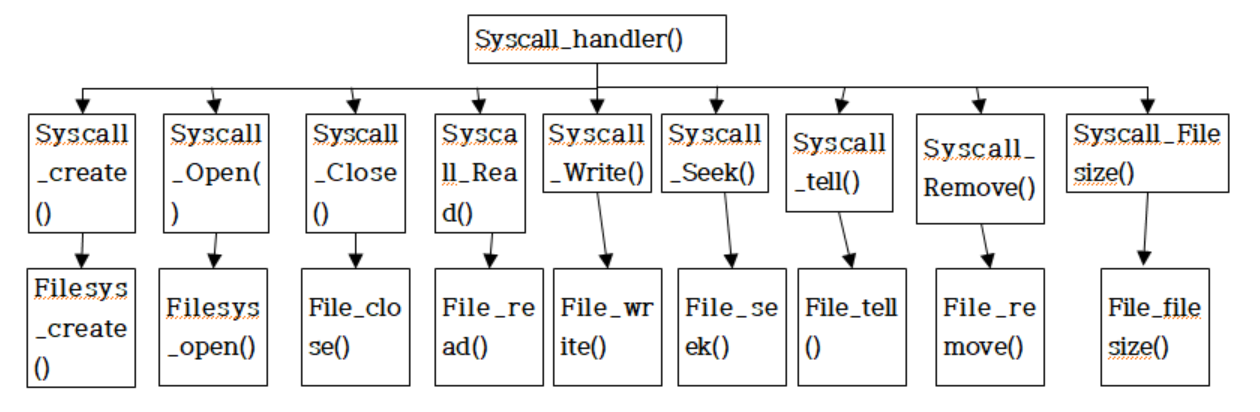


* + - **File close와 open deny 구현.**



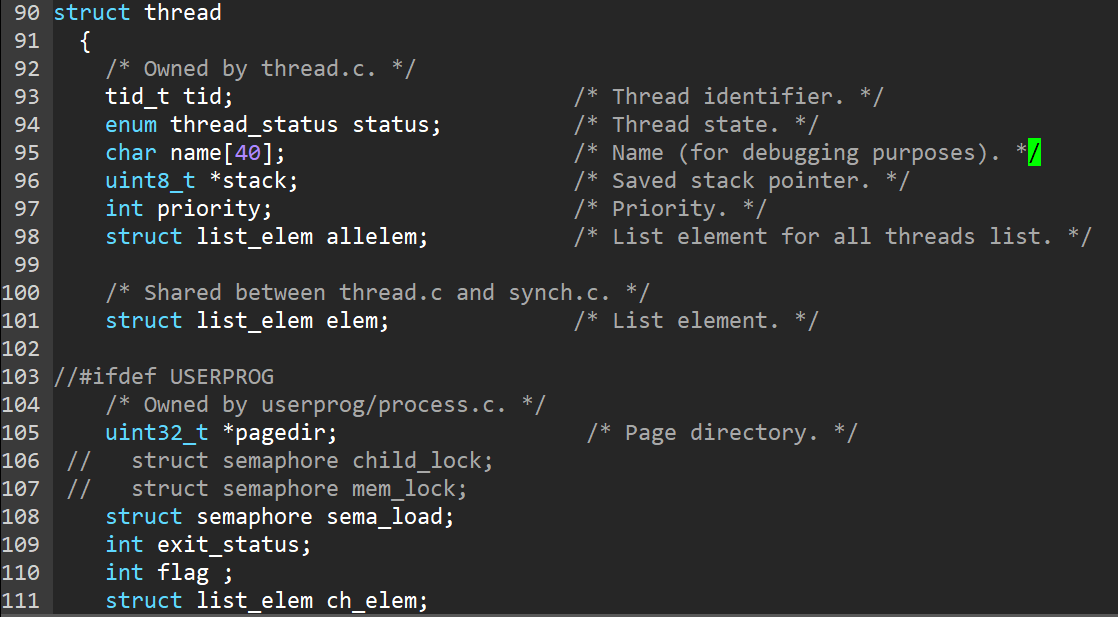
**Start process 에서 로드에 성공했을 경우 즉 파일 생성등에 성공했을 때 락을 걸어 아무도 접근 하지 못하게 한후, 파일 접근을 못하게하는것으로 구현하였다.**

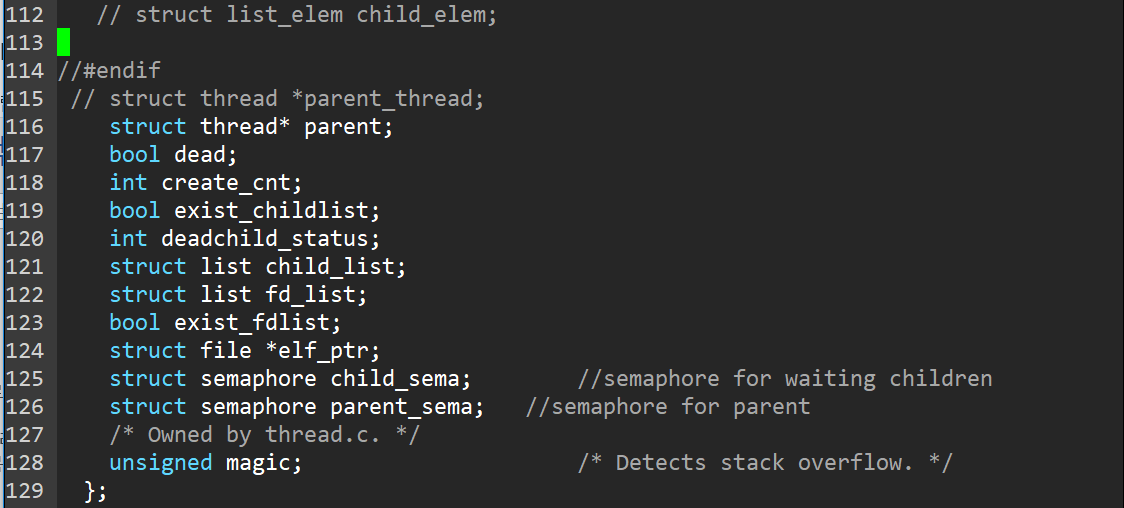
1. **연구 결과**
   1. **합성 내용**



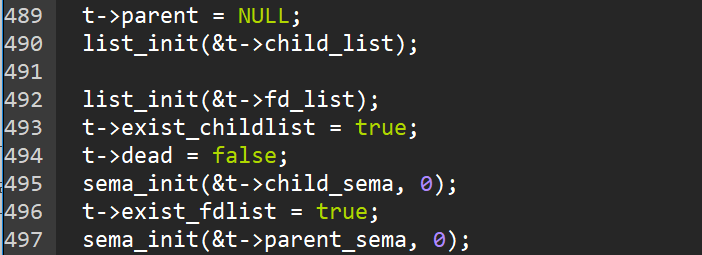
* Syscall\_create함수는 파일의 이름을 파라미터로 받아서 파일을 생성한다.
* Syscall\_remove 함수는 파일의 이름을 파라미터로 받아서 해당하는 파일을 지운다.
* Syscall\_open함수는 파일의 이름을 파라미터로 받아서 파일을 연다. 이때 성공적으로 파일을 연다면 fd 값을 부여받게 된다. 이 fd 값을 이용하여 close, read, write, seek. Tell, filesize 함수를 수행하게 된다. fd값은 항상 0부터 시작하며 0인 경우 STDIN, 1인경우 STDOUT이고 파일이 없을 경우 fd값은 2부터 시작해서 1씩 증가하게 된다.
* Syscall\_close 함수에서 성공적으로 file\_close를 수행하게 되면 현재 thread에서 해당 fd값과 같은 fd값을 가지는 file을 NULL로 설정해준다.
* Synchronization 부분은 file open, read, write시에 수행해준다. 기존에 구현했던

wait과 exit부분에도 synchronization을 구현하기 위해서는 수정한 부분들이 있었다.

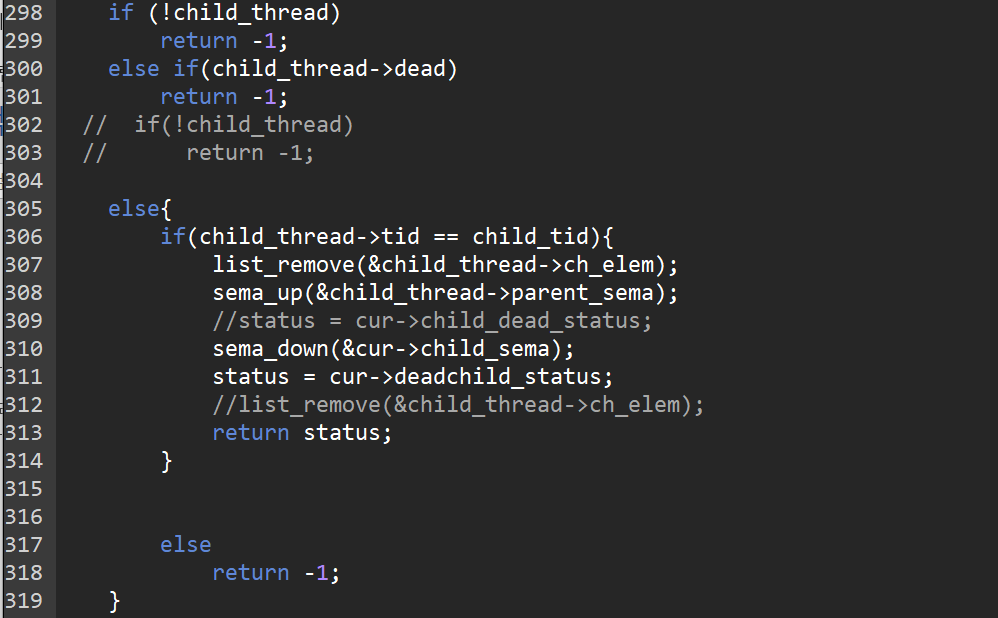
* 1. **제작 내용**
* **Thread**
* 



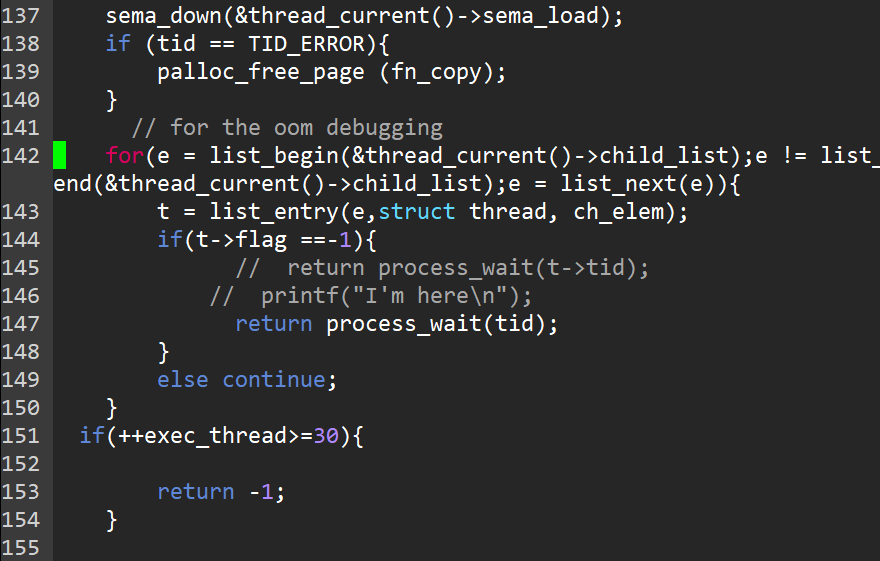
**File\_descripter들을 list로 이어주는 fd\_list를 추가하였다. 또한 그것의 존재 여부를 활용하기 위한 변수인 exist\_fdlist를 추가하였다. Wait 구현 및 동기화를 위한 semaphore인 child\_sema, parent\_sema와 file pointer elf\_ptr을 추가하였다.**



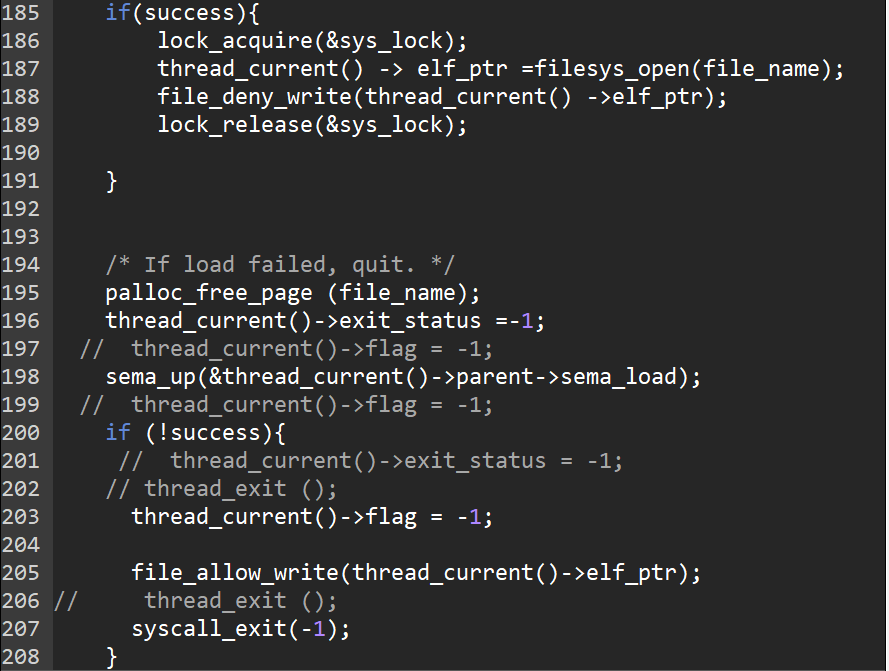
* Thread\_init()에서내에서 sema\_init을 해주어 thread가 하나 생성될 때마다 semaphore를 초기화 하도록 구현 하였다.



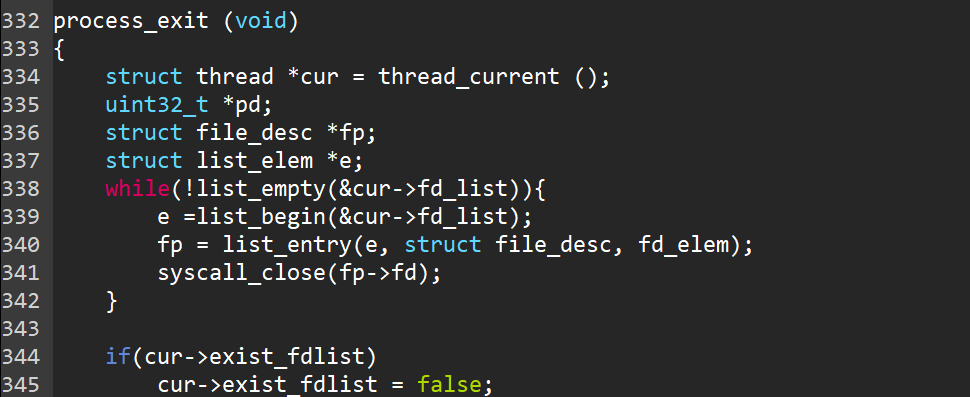
* **Process wait에서는 semaphore를 이용하여 자식 쓰레드가 작업을 끝날 때 까지 부모 쓰레드가 block 되고 그것이 끝나면 풀리게 구현하였다. Busy-waiting 과 같은 원리이고 구현 방법만 다르다.**



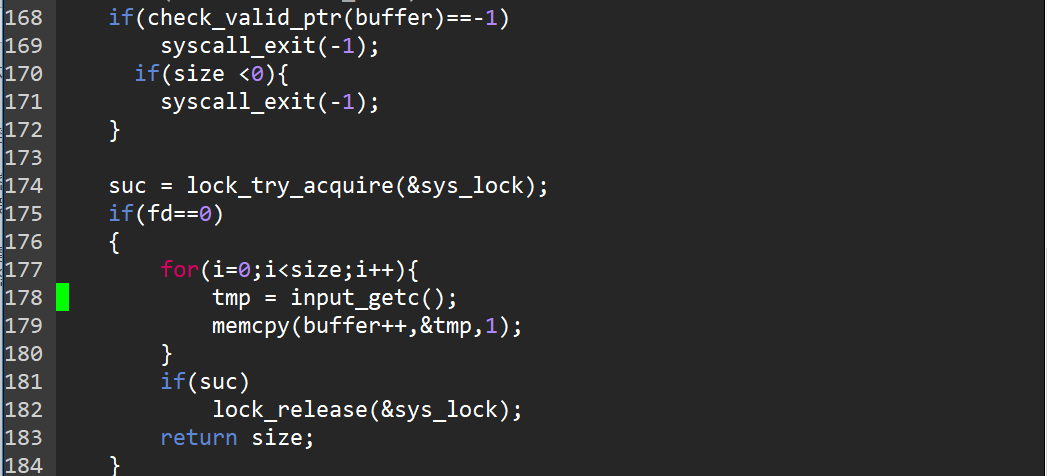
* **Process\_execute에서 동기화 및 Multi-oom을 위해 flag 설정 및 semaphore를 이용한 코드를 작성하였다.**

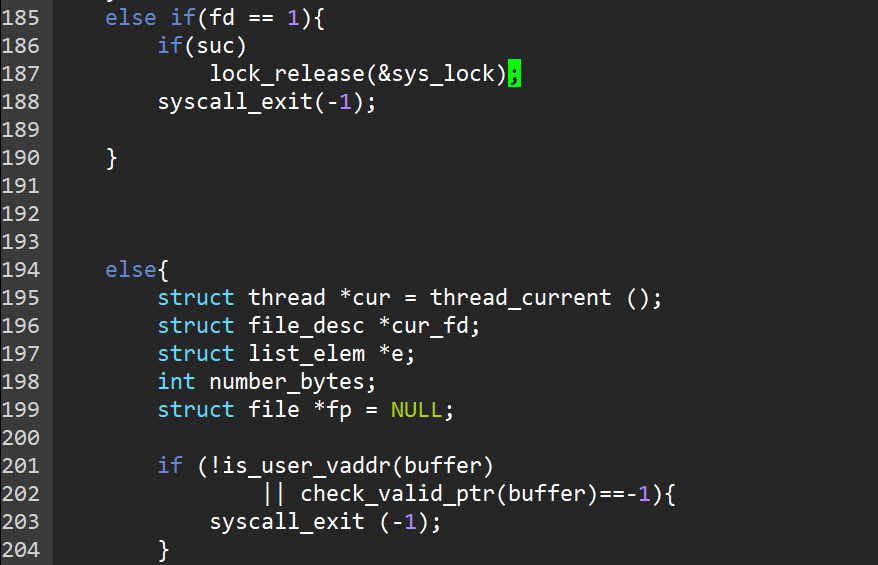


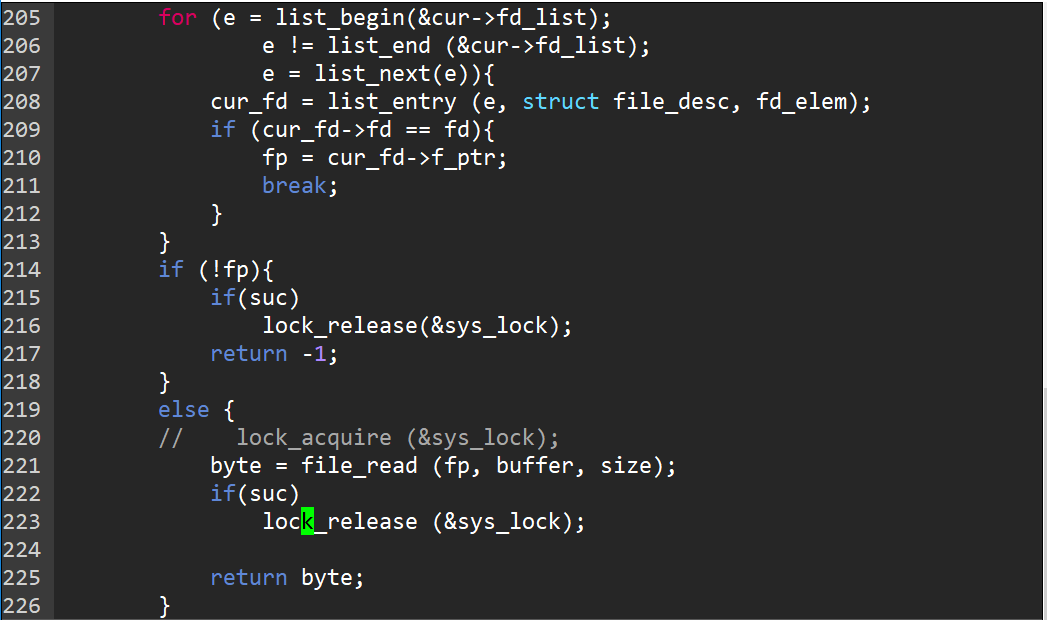
* **Start\_process에서는 load가 실패할 때와 성공할 때를 구분하기 위하여 flag를 설정하였고 동기화 구현을 위하여 파일 포인터 elf를 이용하여 구현하였다.**
* **실패한 경우 sema\_up을 통해 down된 parent를 다시 살려준다.**



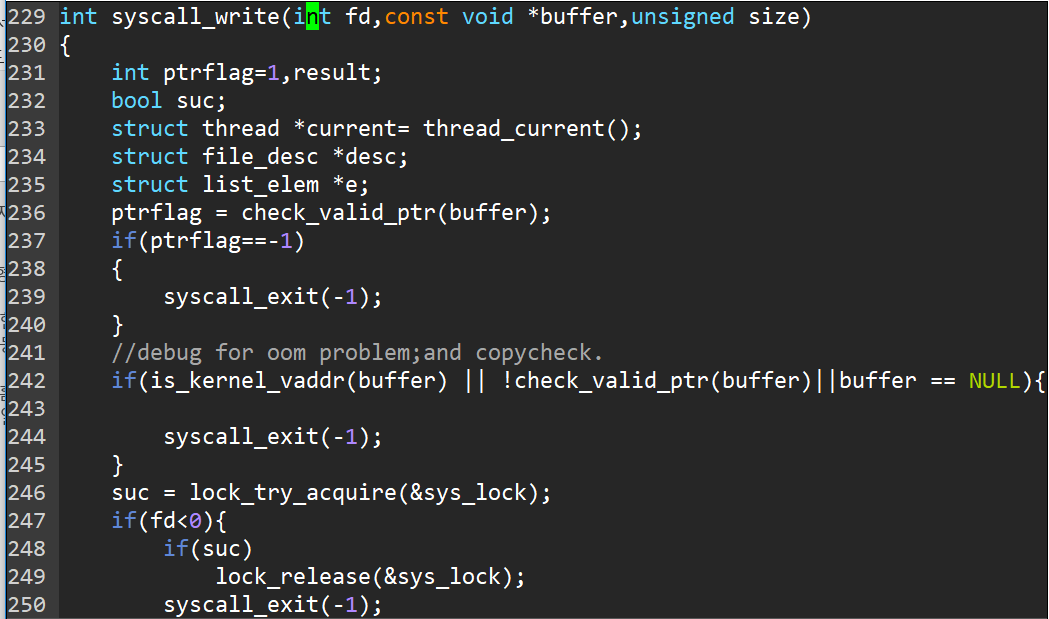
* **Process\_exit에서 나가기 전에 fd가 남아있다면 모두 list에서 제거하고 파일을 닫아준다.**

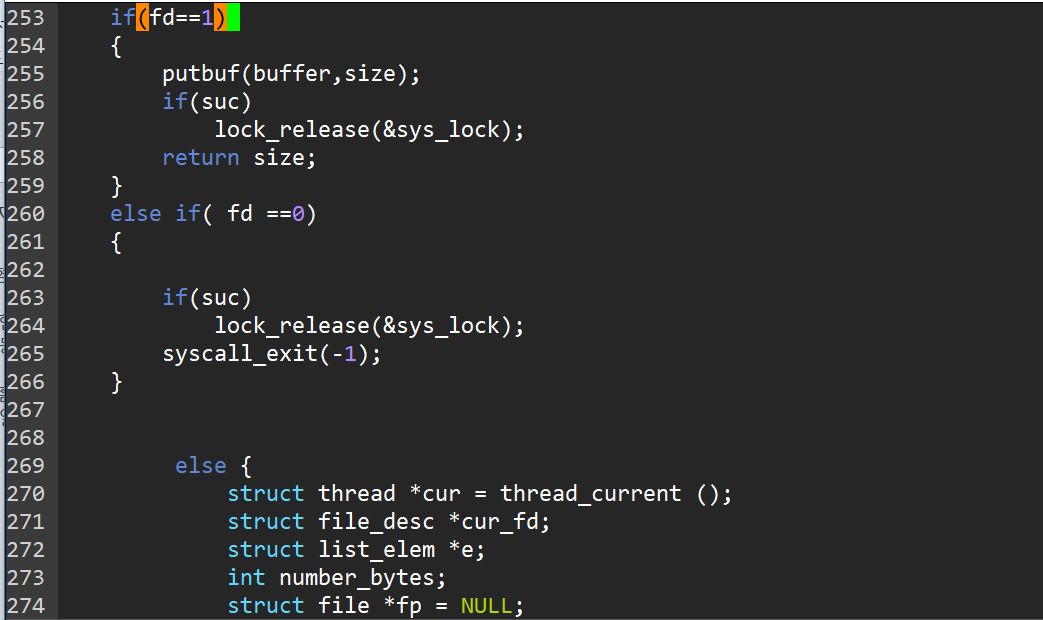


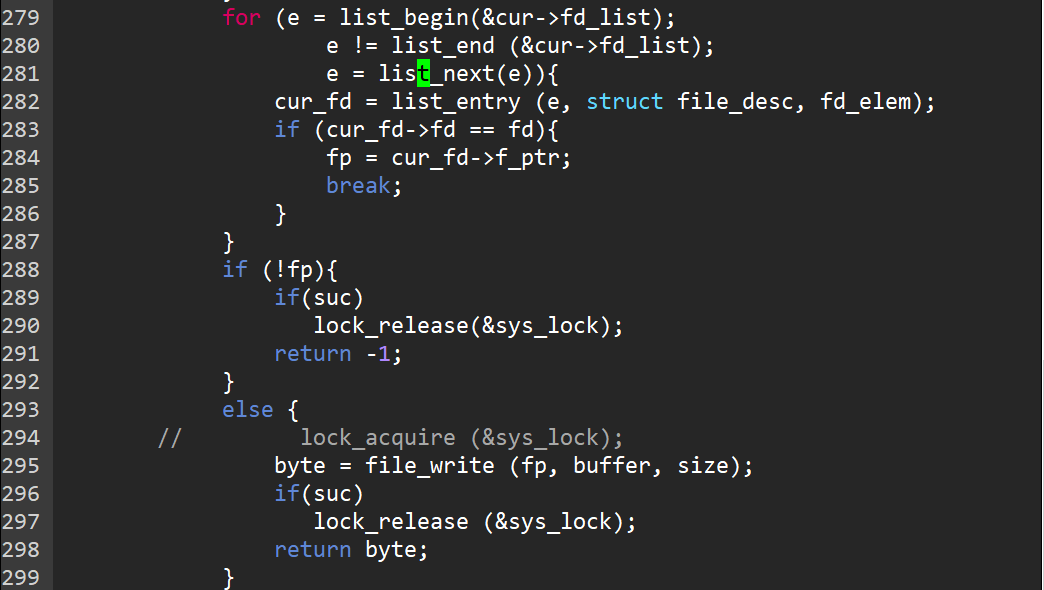




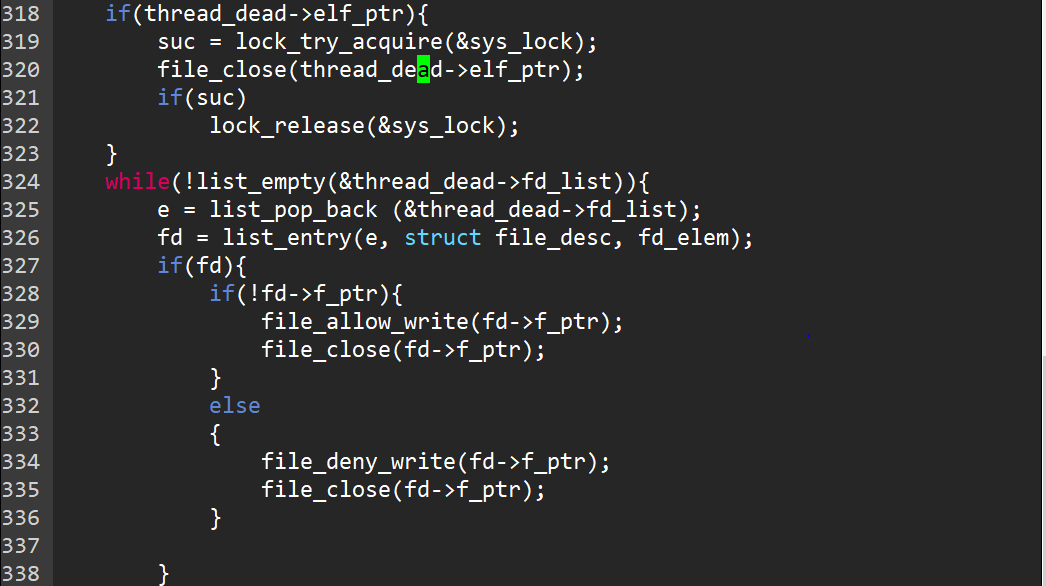
* **Syscall\_read 에서는 fd == 0일때는 STDIN 이므로 읽기를 수행하고 1 일때는 STDOUT이므로 읽지 않고 나머지 fd>0에 대하여 file\_read를 수행한다.**

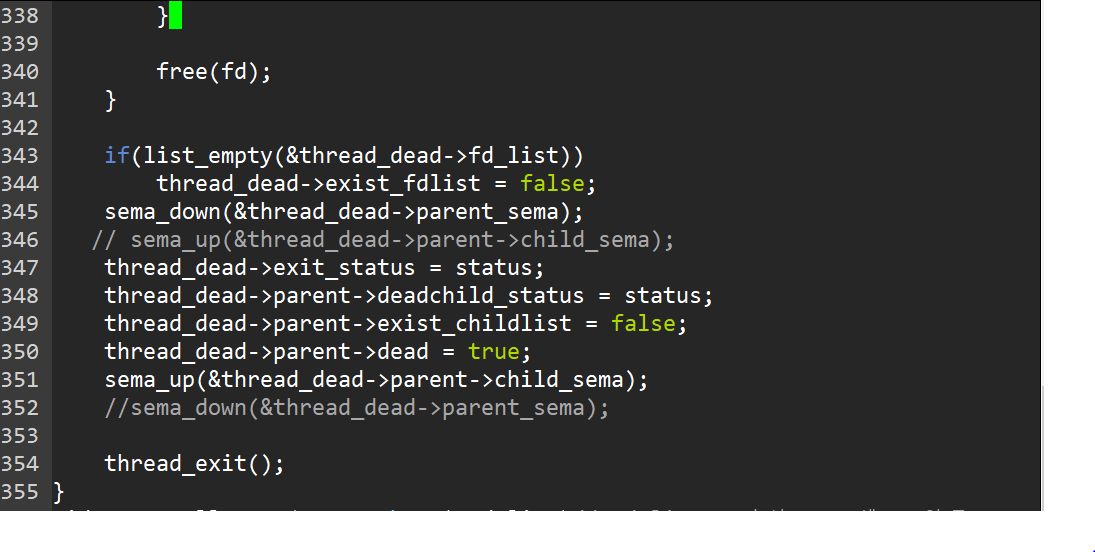




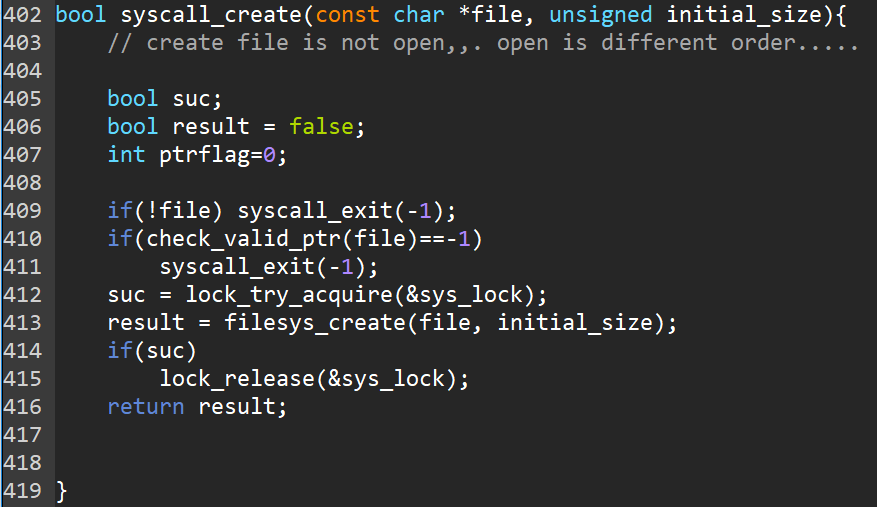


**Syscall\_write에서는 fd<=0 에 대해서는 에러, fd == 1에 대해 STDOUT 에 대한 기능 수행, fd>1에 대해서는 file\_write를 수행한다.**

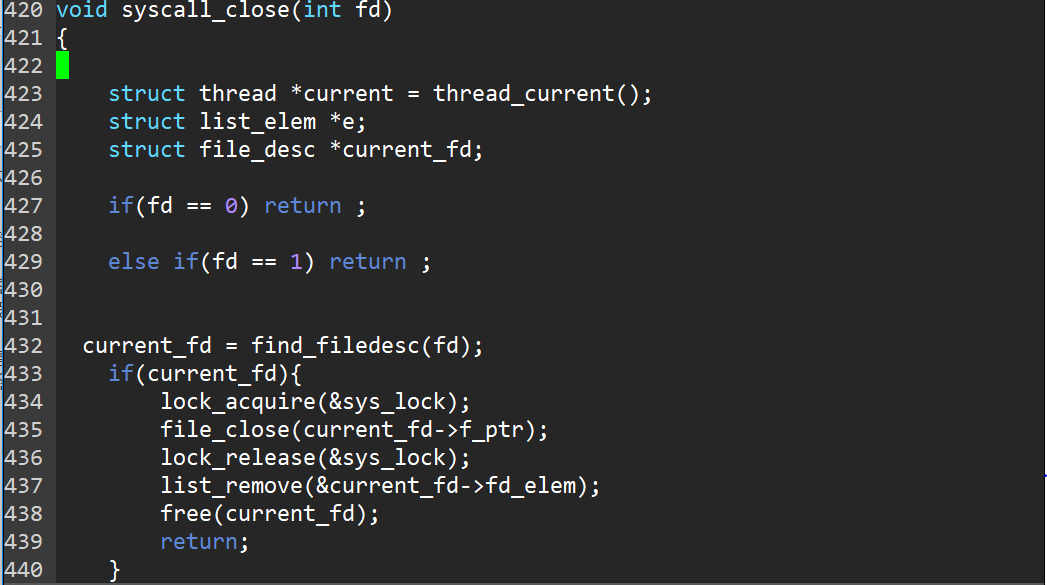




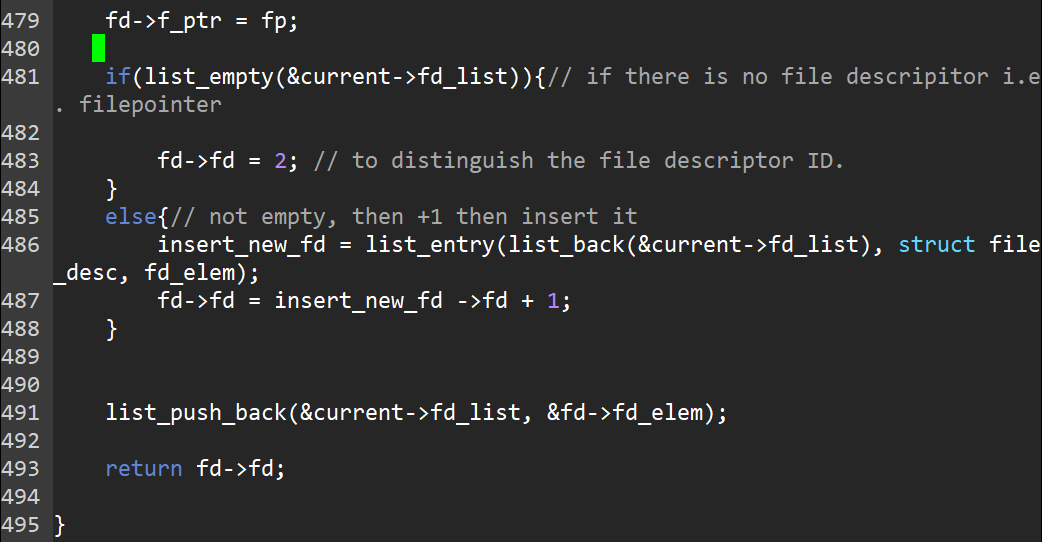
* **Syscall\_exit는 동기화 및 multi-oom 부분 수정을 위해 semaphore와 파일포인터 elf를 이용하였다.**



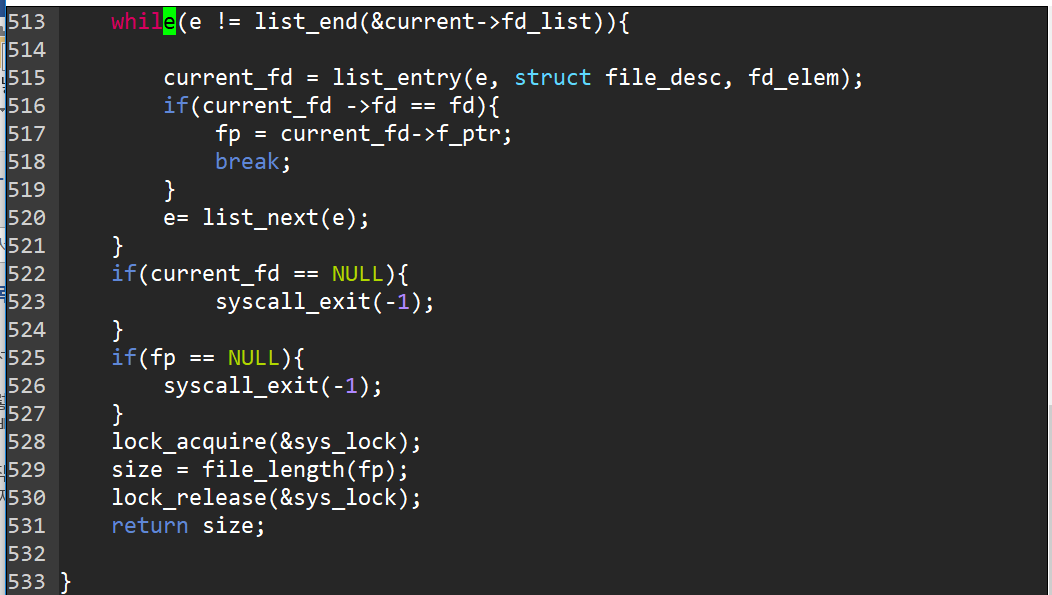
* **Syscall\_create 함수는 파일 create에 성공하면 true, 아니면 false를 리턴한다.**



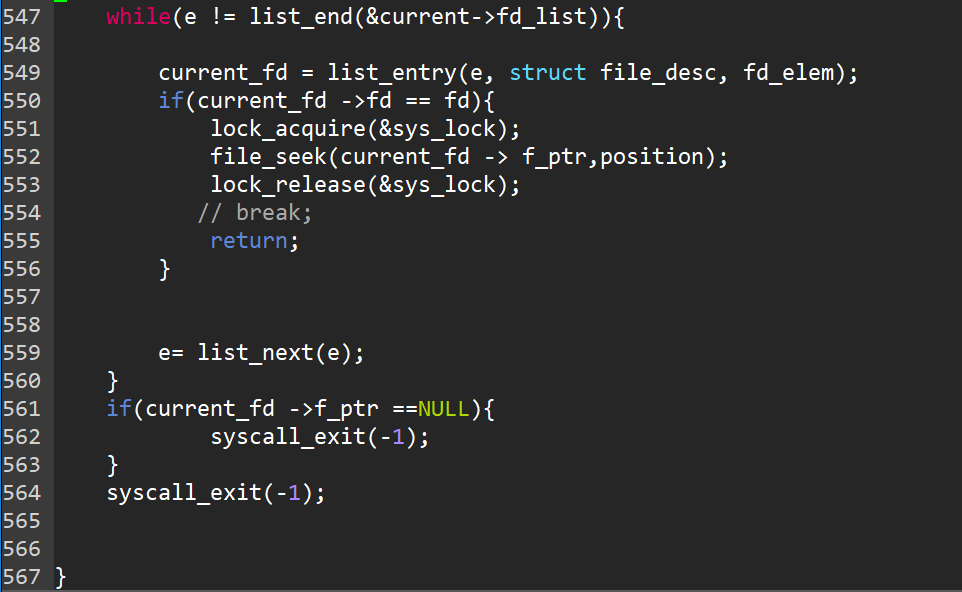
* **Syscall\_close는 해당 fd와 같은 fd 값을 가지는 file\_desc를 찾아서 해당 파일을 close한다.**



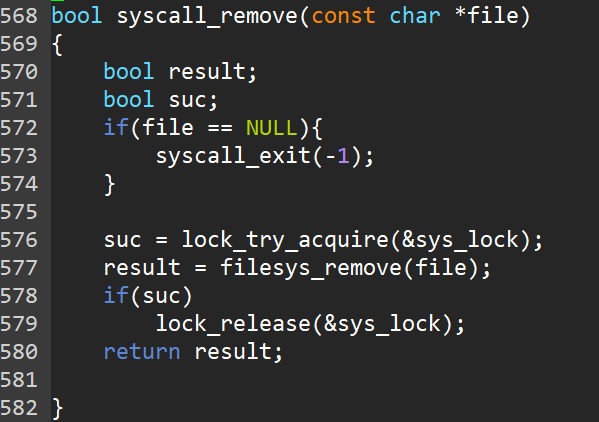
* **Syscall\_open함수는 파일의 이름을 파라미터로 받아서 파일을 연다. 이때 성공적으로 파일을 연다면 fd 값을 부여받게 된다. 이 fd 값을 이용하여 close, read, write, seek. Tell, filesize 함수를 수행하게 된다. fd값은 항상 0부터 시작하며 0인 경우 STDIN, 1인경우 STDOUT이고 파일이 없을 경우 fd값은 2부터 시작해서 1씩 증가하게 된다.**



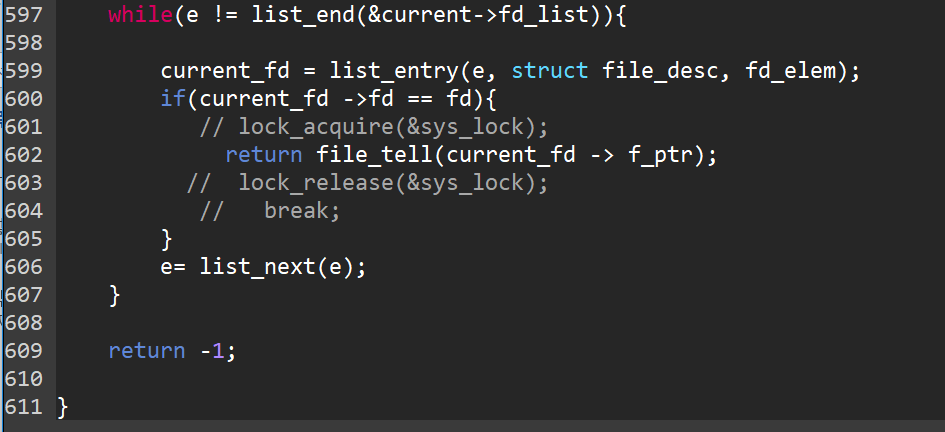
* **Syscall\_filesize 함수는 파일이 비어있지않다면 file\_length를 이용하여 파일의 사이즈를 리턴한다.**



* **Syscall\_seek 역시 file\_seek 함수를 호출해 기능을 수행한다. file안에서 file의 시작부분부터 new\_pos까지 현재 위치를 set하고 다음 byte를 바꾼다.**



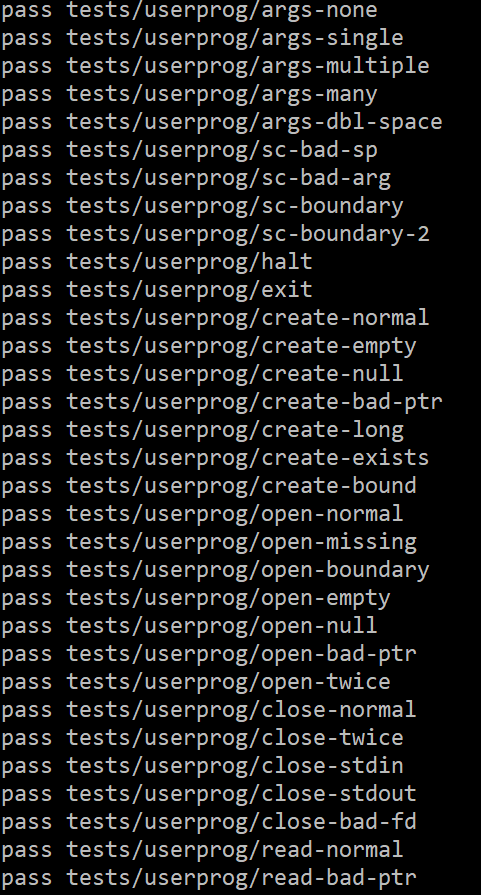
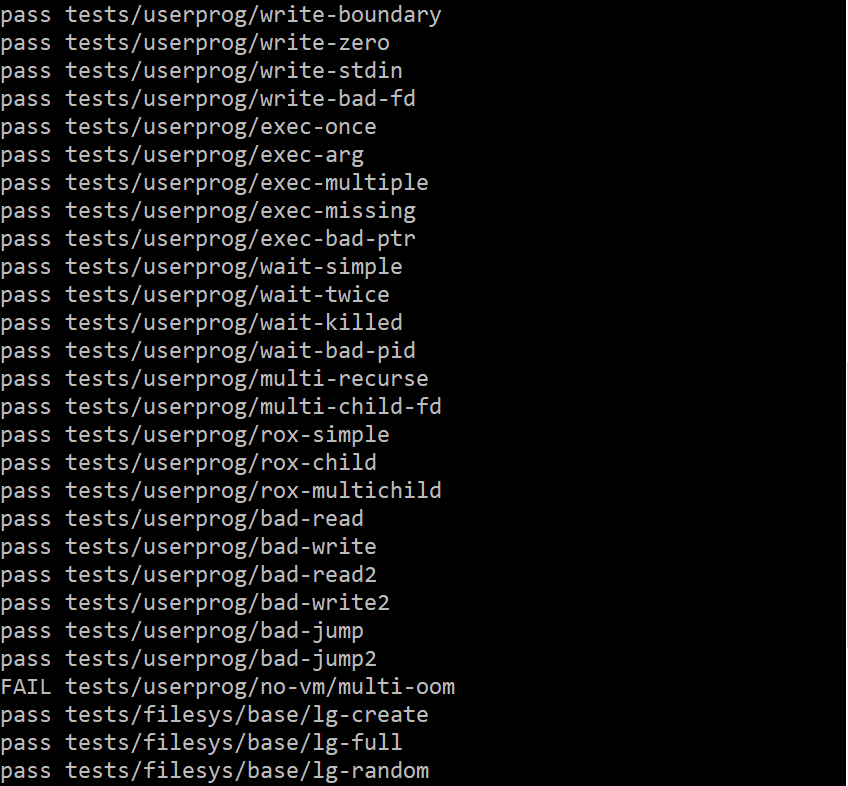
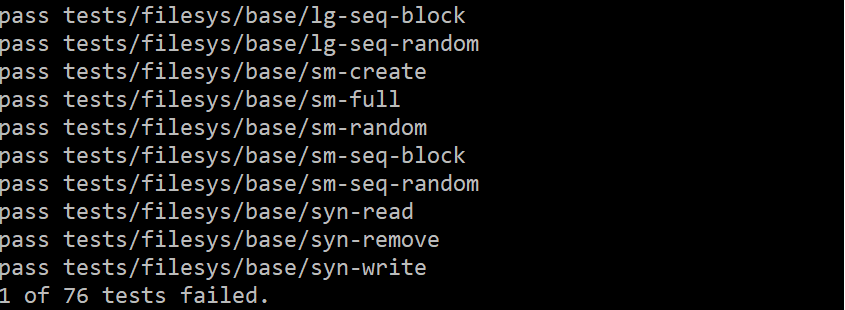
**-syscall\_remove 함수는 문자열을 받아와 같은 이름을 가진 파일을 삭제한다.**



**-syscall\_tell 함수는 fd에 해당하는 file open을 하고 read하거나 write한 다음 byte의 position을 return 한다.**

* 1. **시험 및 평가 내용**

**테스트 케이스에 관한 이미지 첨부. ( make check)**

****

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.**
* **황순 – 운영체제 프로젝트를 하면서 수업시간에 배운 semaphore 나 lock의 개념에 대해서 직접 코딩을 하면서 개념을 익힐 수 있었다. Multi – oom을 많은 노력 끝에도 성공하지 못해서 아쉬웠고, 이를 통해서 밑그림을 좀 더 확실히 그리고 코딩을 해야겠다고 생각했다. 또한 운영체제 지식을 잘 아는 것 뿐만 아니라 팀 프로젝트이므로 팀원끼리 의사소통이 잘 되는 것이 이번에도 정말 중요했다고 느꼈다.**