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

**(설계 프로젝트 수행 결과)**

과목 명 : [CSE4070] 운영체제

담당 교수 : 김영재 교수님

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

개발 기간 : 2019/11/1 – 2019/11/17

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

**제출일 : 2019년 11월 17일**

**참여 조원 : 이정원, 최아연**

1. **개발 목표**

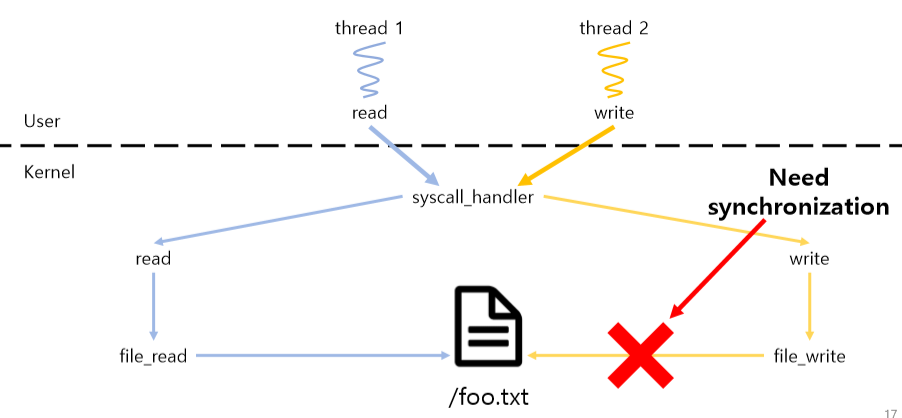
* Project1에서 구현한 코드를 수정해 파일 시스템 관련 시스템콜을 완성한다. Project1에선 STDIN, STDOUT에 대해서만 구현했지만, 이를 확장한다**.**
* 또한 project1에서 busy waiting으로 구현한 임계구역문제를 Lock과 Semaphore로 수정한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* **Files System Call**
* Project1에서 STDIN과 STDOUT일 경우에 대해서만 구현했으므로, 이를 확장한다.
* 이를 위해 system call(create, remove, open, close, filesize, read, write, seek, tell)을 추가구현 혹은 수정한다.
* **Synchronization**
* 이전에 busy waiting으로 구현한 critical section 문제를 semaphore와 lock을 이용해 구현한다.
* Semaphore 구조체와 lock구조체를 이용한다. 또한 sema\_wait, sema\_exit, sema\_load 구조체를 추가선언해 활용한다.
  1. **개발 내용**
* **File system call**



* 위는 pintos에서 file system call을 실행하는 예시이다.
* Sample.txt라는 파일을 열기 위해선 user단계에서 open 함수를 호출해, kernel 단계에 진입한다. Filesys\_open 함수는 커널 단계에서 file open 작업을 실행한다.
* **Critical Section**
* 둘이상의 프로세스, 쓰레드가 동시에 접근해서는 안되는 shared data를 접근하는 코드의 일부분을 말한다.



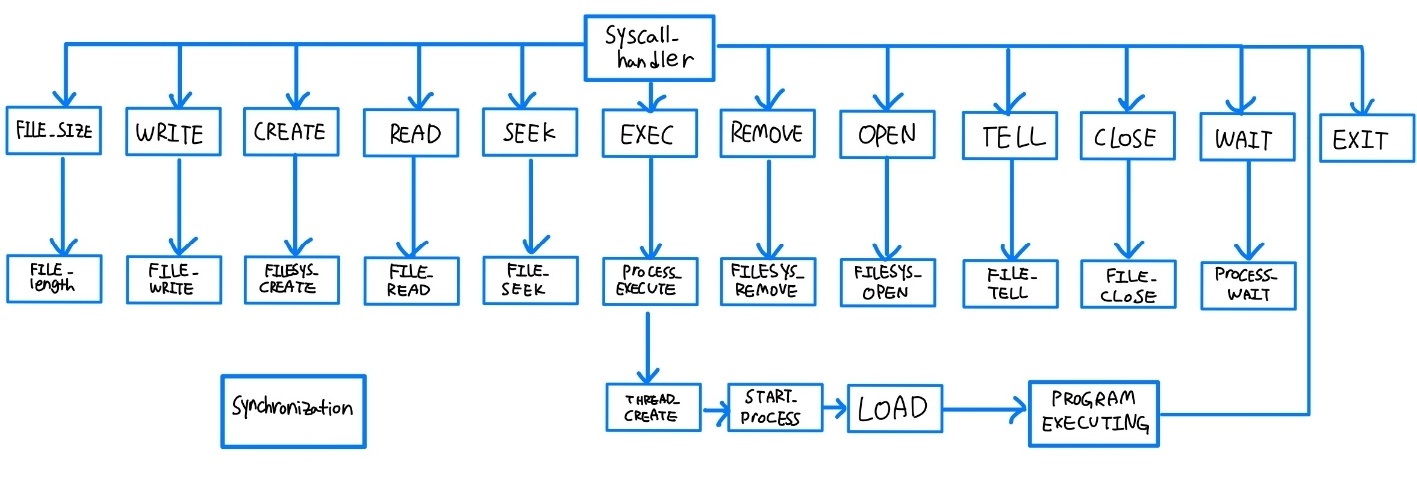
* 위의 경우에선 foo.txt를 read가 접근 중일 때 write가 접근하면 안된다. 따라서 프로세스가 임계구역에 들어가고자 할 때 지정된 시간만큼을 대기해야 한다.
* 이를 위해 다양한 기법이 사용되는 데, 이번 project에선 semaphore기법을 활용했다.
* **Semaphore**
* 복수의 작업을 동시에 병행하여 수행하는 프로세스에서 공유 자원에 대한 접속을 제어하기 위하여 사용되는 신호이다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

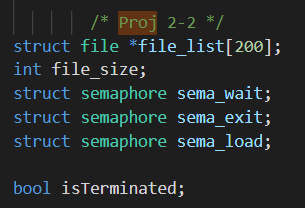
* 2019년 11월 13일 : 코드 분석 및 구현해야할 항목 파악
* 2019년 11월 14일 : 전체 구현
* 2019년 11월 15일 ~ 2019년 11월 16일 : 보고서 작성
  1. **개발 방법**
* **Files System Call**
* Userprog/syscall.c에서 create 부분을 수정한다.
* Create, remove, open, filesize, seek, tell, close부분을 구현한다.
* 프로세스가 열었던 파일을 자동으로 close하지 않으므로 직접 close를 수행한다.
* Null pointer를 open하려는 경우도 있으므로, 이를 해결하기 위해 예외처리를 한다.
* **Synchronization**
* Executable of running thread에 write하는 경우를 막아야한다.
* Open할 때 thread\_name과 비교해서 일치하면, write하지 못하도록 이를 확인한다.
* File\_deny\_write, file\_allow\_write 함수를 적절히 활용해 이를 구현한다.
* Thread.h에 semaphore 구조체 sema\_wait, sema\_exit, sema\_load를 선언하고, 이를 활용해 userprog/process.c에서 busy waiting으로 구현한 critical section문제를 semaphore 기법으로 바꾼다.
  1. **연구원 역할 분담**
* 이정원 : 프로젝트 구현
* 최아연 : 보고서 작성

1. **연구 결과**
   1. **합성 내용**

* Pintos 흐름도



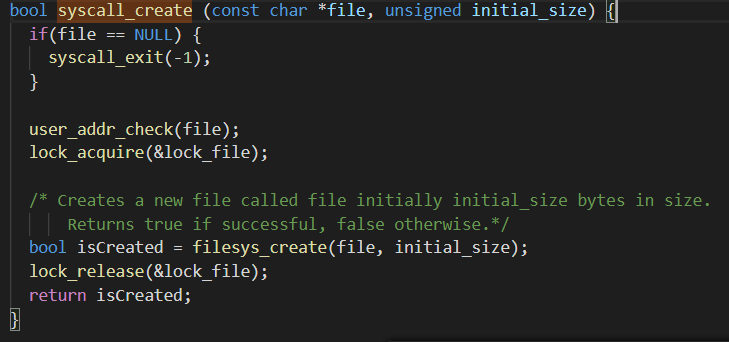
* 1. **제작 내용**
* **Thread.h**

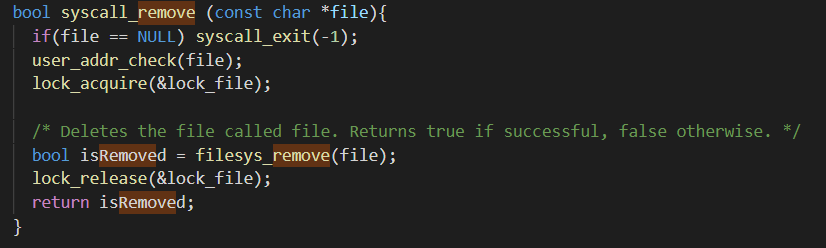


Stdin(0), stdout(1)외에 다른 fd값을 받아 파일 입출력을 실행하므로, file\_list 배열을 선언해 주고 syscall에서 활용한다.  
syscall에선 fd를 매개변수로 받아와, fd번째의 파일 포인터를 찾는 연산을 수행한다.  
선언해준 semaphore 구조체들은 process.c에서 쓰인다.

* **Syscall**

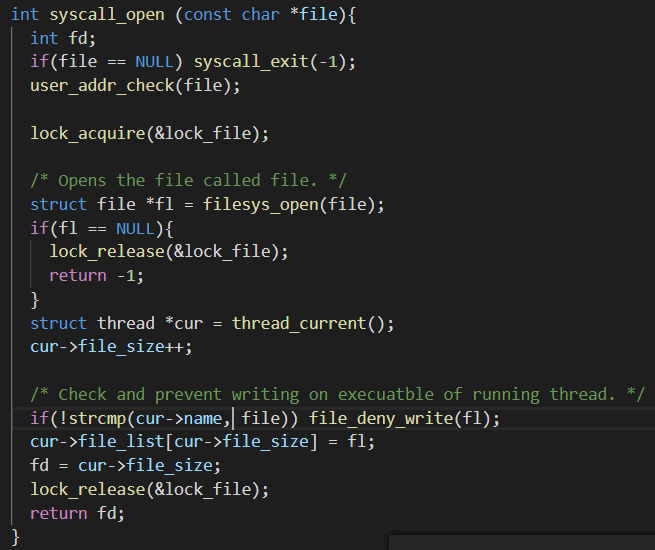
Critical section 문제를 해결하기 위해 user\_addr\_check와 lock을 이용했다.  
이 부분은 후에 synchronization 구현에서 설명한다..

* **Create**
*   
  매개변수로 받은 이름과 파일크기로 파일을 생성하고, 결과값을 return한다.
* **Remove**



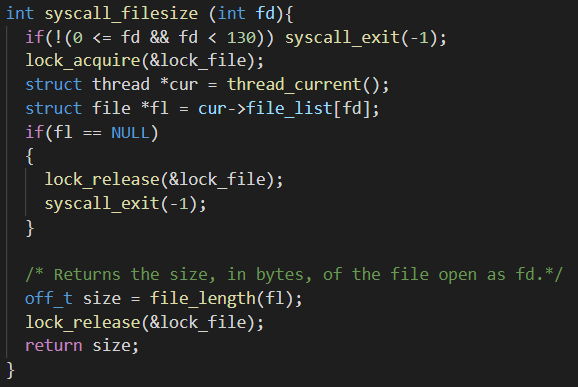
매개변수로 받은 파일이름을 filesys\_remove로 제거하고, 결과값을 return한다.

* **Open**



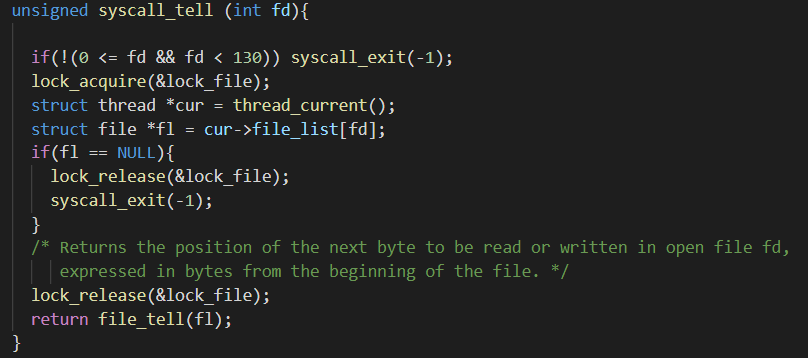
filesys\_open함수를 통해, 파일 포인터를 얻는다.   
그 후 실행중인 쓰레드에 write 작업이 수행되는 것을 막기 위해 파일이 실행 중인지 확인한다.

* **Filesize**



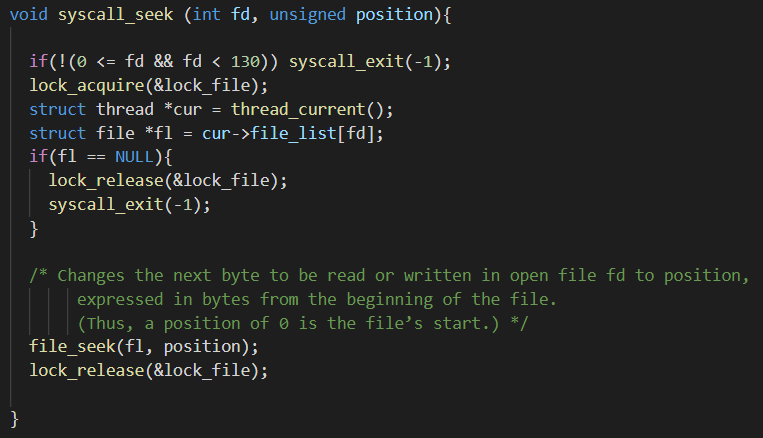
현재 thread에 저장된 file\_list를 탐색하며, fd번째 file의 크기를 return해주는 함수이다.  
Fd에 해당하는 파일 포인터를 구한 후 file\_lenght함수를 통해 파일의 크기를 구한 후 return한다.

* **Tell**



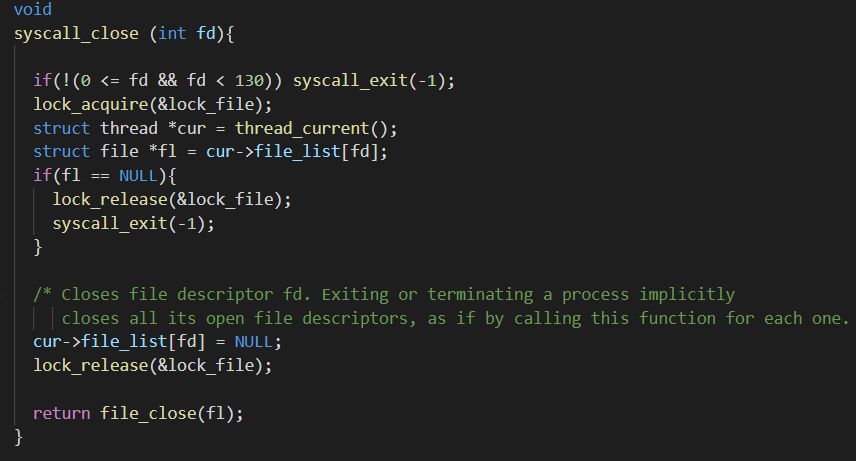
현재 thread의 fd번째 file\_list를 탐색해 포인터를 찾는다.   
그 후 file\_tell에 파일 포인터를 전달하여, 함수의 position을 구하고, return한다.

* **Seek**



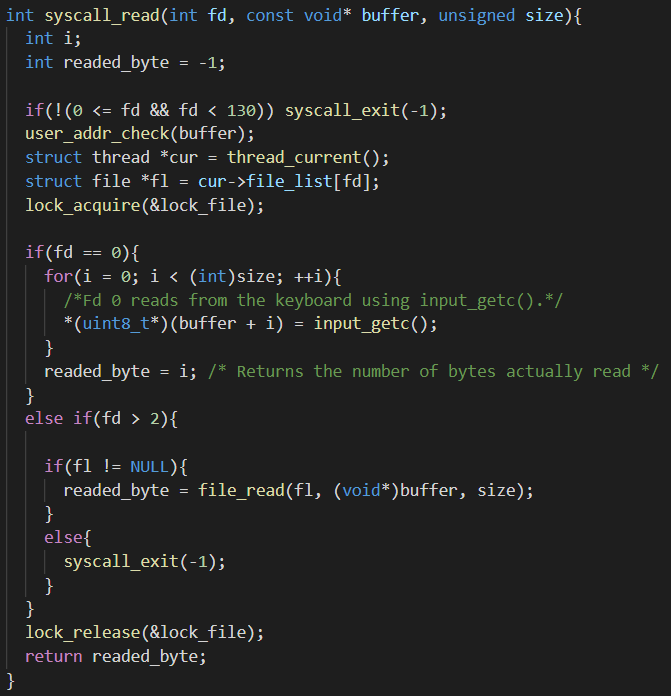
현재 실행중인 thread의 file\_list를 탐색해 fd의 파일 포인터를 찾는다.   
그리고 이 파일 포인터와, position을 통해 파일 포인터의 position을 수정한다.

* **Close**



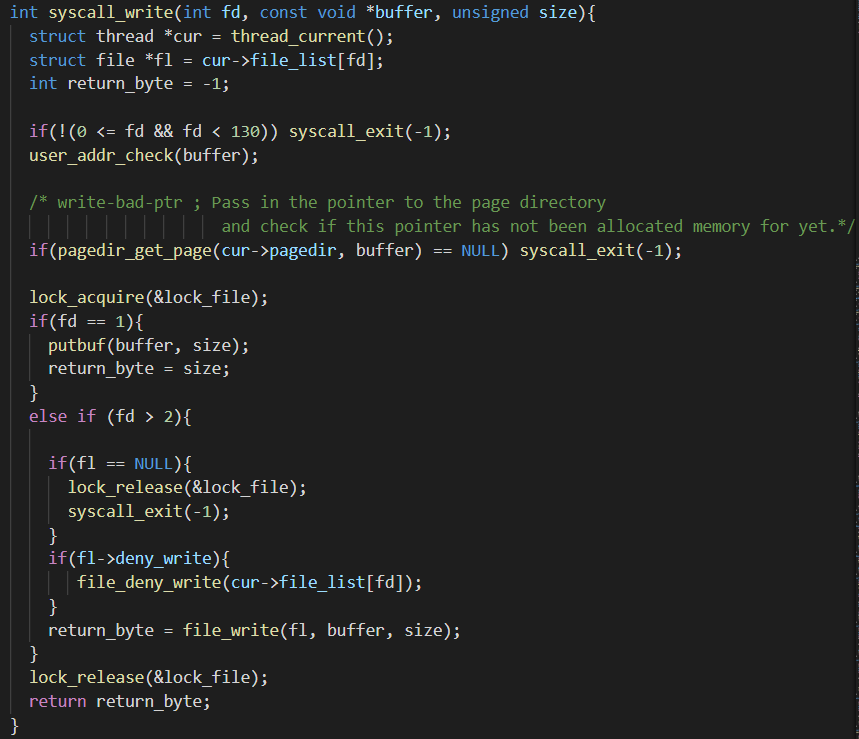
실행중인 thread의 fd번째 file\_list를 탐색해, 해당 파일의 파일포인터를 찾는다.  
그 후 file\_close()함수를 통해 해당 파일을 삭제하고 성공여부를 return한다.

* **Read**



현재 thread의 fd번째 file\_list를 탐색해 파일포인터를 찾는다.  
만약 fd가 0일 경우는 standard input인 경우인데, 이때 input\_getc()를 활용해 입력을 받는다.  
fd가 0이 아닌 경우는 standard input이 아닐 경우인데, 이 때 탐색해둔 파일터가 NULL이 아닐 경우에 file\_read함수를 호출하여, read작업을 수행한다.   
그리고 file\_read의 return값을 return한다.

* **Write**

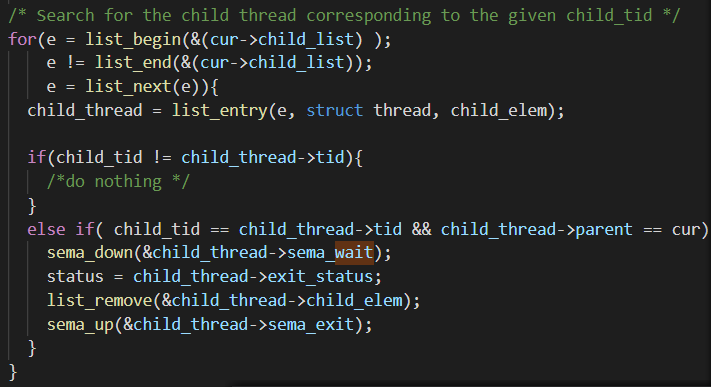


실행중인 thread에서 fd에 해당하는 파일 포인터를 찾는다.   
만약 fd가 1이라면 standard output 이므로 putbuf()함수를 활용해 write작업을 수행한다.  
fd가 2 이상일 경우, standard output이 아니다.  
이 경우에는 파일 포인터가 null이 아니고, deny\_write가 null이 아닐 때 (critical section문제 확인용) file\_write()함수를 실행한다.  
그 후 file\_write의 return값을 return한다.

* **Process.c**

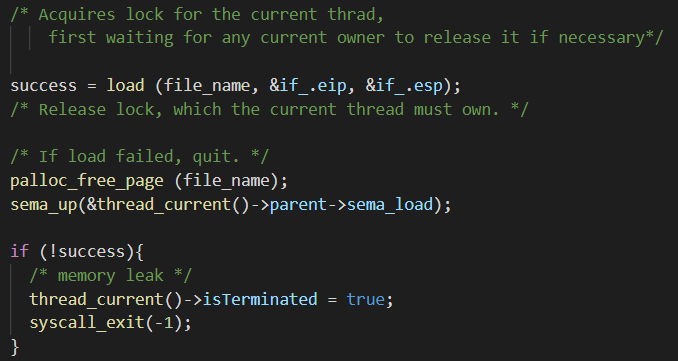
자식 thread가 load되기 전에 부모 thread가 종료되어 버린 경우를 처리해준다**.**

* **Process\_wait**



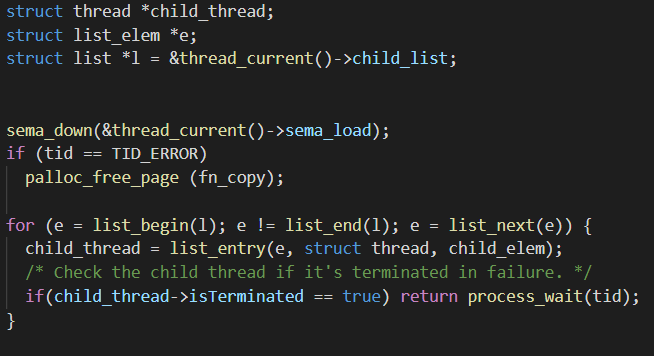
주어진 child의 tid와 일치하는 child thread를 탐색한다.  
만약 일치하는 child thread가 있을 경우, 부모 쓰레드는 자식 쓰레드가 종료될 때까지 wait 작업을 수행해야한다.  
thread.h에 선언해둔 sema함수들을 활용해 작업을 수행한다.

* **Start\_process**

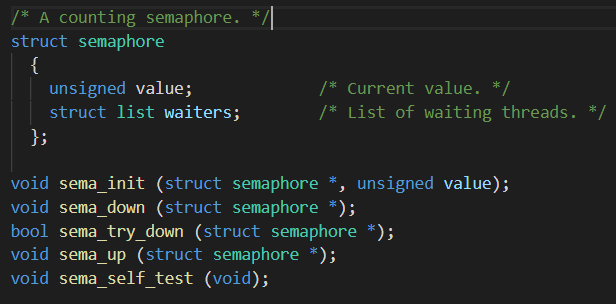


Interrupt로 프로세스를 실행시킨 후, 현재 thread에 lock을 걸어, release가 호출될 때까지 대기한다.   
thread가 필요할 때, lock을 해제하고, 만약 load에 실패했을 경우 종료한다.

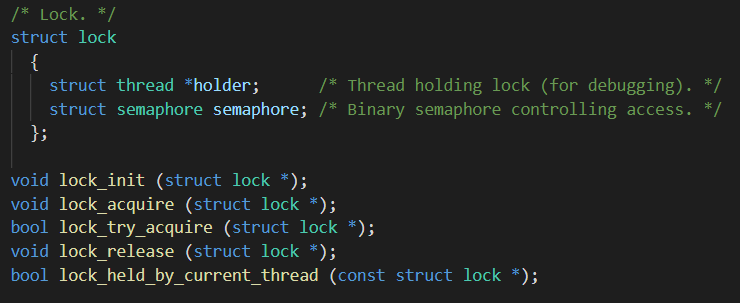
* **Process\_Execute**

load에 실패했을 때의 처리를 해준다.  
isTerminated가 true라면, chil\_threa가 강제종료 된 것이므로, 이를 확인하여, process\_wait로 실패한 process를 회수한다.

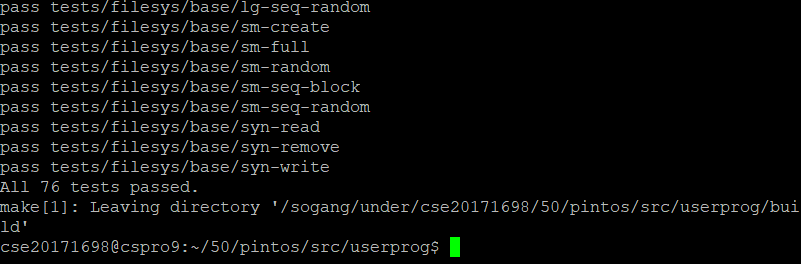
* **Synchronization**
* **Semaphore**



* Semaphore 구조체: 현재 sema값을 저장하는 value와 대기중인 thread list 두개 변수로 이루어져 있다.
* **Lock**



* Lock 구조체: 현재 lock 중인 thread와 그 thread의 semaphore를 나타내는 semaphore 구조체로 이루어져 있다.
* 한 thread가 임계구역에 진입하면 lock\_acquire을 호출해, lock을 걸고 그 thread가 임계구역에서 빠져나오면, lock\_relaese를 호출해 lock을 해제한다.
  1. **시험 및 평가 내용**
* **Test cases 수행결과**



* 76개의 test 항목에 대해서 all pass

1. **기타**
   1. **연구 조원 기여도**

* 이정원: 50%
* 최아연: 50%
  1. **소감**
* Pintos환경 상에서 파일 시스템 관련 시스템 콜을 완성하면서, OS가 어떻게 파일을 처리하는지 원리에 대해 공부할 수 있었다. 뿐만 아니라 linux환경에서 project를 수행하며, Project1에서는 몰랐던 linux환경에 대해서도 공부할 수 있었다.
* 계속 kernel panic 오류가 생겼다. 해결 방법을 찾기 위해, 다른 조들과 토론도 해보고, 인터넷으로 여러 자료들을 찾아보았다. 오류를 해결하기 위해 계속 고민하고, 코드를 찬찬히 분석해 결국 오류를 해결했을 때 뿌듯함을 느꼈다.
* 코드를 조금 수정했을 뿐인데, 오류가 나는 경우가 종종 있었다. 모든 코드가 서로 유기적으로 연결 되어있고, 사소하게 보인 함수가, 중요한 역할을 할 수도 있다는 것을 다시한번 깨닫았다. 따라서 변수명과 함수명을 지을 때, 직관적으로 무엇을 담당하는 변수인지 함수인지 알 수 있게 짓는 것이 중요하다고 느꼈다.
* 저번과 마찬가지로 팀 프로젝트였기 때문에, 가독성 좋은 코드를 짜는 것이 중요했다. 팀프로젝트를 하며, 다른 사람도 쉽게 이해할 수 있는, 개발자에 맞는 코딩 스타일을 익히게 되었다.