**Pintos Project 2 : User Program**

**(설계 프로젝트 수행 결과)**

과목 명 : OS

담당 교수 : 김영재 교수님

조 / 조원 : 11조 / 장지훈 안시현

개발 기간 : 2018/10/20~11/14

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

**제출일 : 2018/11/14**

**참여 조원 : 안시현/장지훈**

1. **개발 목표**

* **File System 관련 System call을 구현한다**
* **create, remove, open, close, filesize, read, write, seek, tell**
* **동기화 방식을 busy-waiting 기반에서 semaphore 기반으로 바꾼다.**

1. **개발 범위 및 내용**
   1. **개발 범위**

* **지난 Project인 User program basic에서 완성하지 못한 file system 관련 system call들을 구현한다**
* **Wait 방식을 busy-waiting 기법에서 semaphore 기법으로 변경한다.**
* **File System 관련 create , filesize, remove, seek, tell 관련 함수는 구현 되어 있으므로 이를 적절히 이용하여 user program과 system call을 연결해 줄 수 있는 함수들을 구현한다.**
* **적절한 동기화를 통해 critical section을 보호하여 각 함수들의 필요한 구역에 적절한 동기화가 이루어 질 수 있도록 한다.**
  1. **개발 내용**

**이번 Project 에서는 앞선 Project에서 구현하지 않았던 File System 관련 system call을 구현할 것이다. 또한 busy-waiting 기법을 semaphore 기법으로 변경할 것이고 각각에 함수들이 적절히 동기화 될 수 있도록 critical section으로 함수 내부의 영역이 보호 될 수 있도록 할 것이다.**

* **Thread Synchronization**

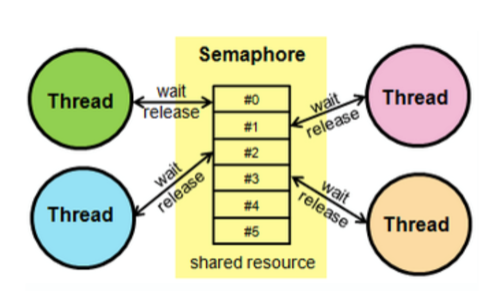
**여러 thread가 하나의 data에 동시에 접근 하려고 할 때를 Race Condition이라고 한다. 이때 data의 최종 연산 결과는 그 data를 마지막에 다룬 process(thread)에 따라 달라진다. 동시에 접근했다고 하더라도 연산의 수행이 더 늦게 끝나는 thread의 결과값을 따르게 된다.**

**즉, 공유 데이터의 동시접근은 data의 불일치 문제를 발생 시킬 수 있다.**

**이러한 data inconsistency를 극복해 주는 것이 thread synchronization의 주 목표라고 할 수 있다. 이번 project에서는 shared data를 수정하려 할 때 그 부분을 critical section으로 보호함으로써 그 부분의 연산을 atomic하게 만들어 주어 data inconsistency를 극복할 것이다.**

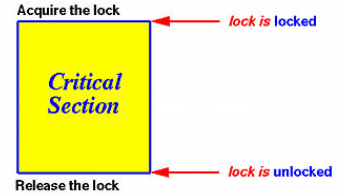
* **Semaphore**

**세마포어(semaphore)는 교착 상태에 대한 해법으로 두개의 Atomic한 함수로 제어되는 정수 변수로 멀티프로그래밍 환경에서 공유자원에 대한 접근 제어를 하는 방식이다. 1개의 공유되는 자원에 제한된 개수의 프로세스, 또는 스레드만 접근할 수 있도록 한다.**



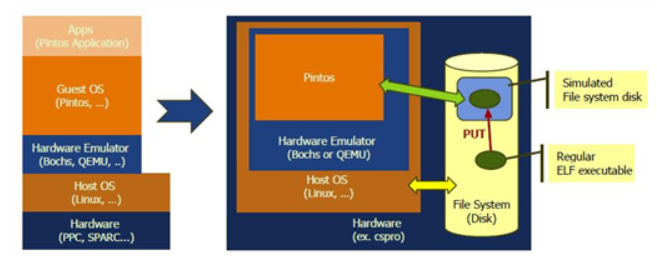
* **Critical Section**

**임계 구역(critical section) 또는 둘 이상의 스레드가 동시에 접근해서는 안되는 공유 자원을 접근하는 코드의 일부를 말한다. 임계 구역은 지정된 시간이 지난 후 종료된다. 때문에 어떤 thread가 임계 구역에 들어가고자 한다면 지정된 시간만큼 대기해야 한다. thread가 공유자원의 사용을 보장받기 위해서 임계 구역에 들어가거나 나올때는 semaphore 같은 동기화 매커니즘이 사용된다**

**.**

* **File System in pintos**

**syscall.c, thread.c, process.c, file.c, filesys.c의 코드가 이 프로젝트의 핵심 코드이며 이 안에서 정의된 함수들을 기반으로 file system 관련 system call들을 구현하였다. 다음은 File system 구성도이다.**

* 

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* **10/20~10/29 : 매뉴얼 이해 및 syscall 함수 작성**
* **10/30 ~ 11/5 : busy\_waiting -> semaphore로 변경**
* **11/5 ~ 11/12 : rox와 synch, multi-oom 테스트 구현 및 수정**
* **11/12 ~ 11/14: 최종 코드 정리 및 보고서 작성**
  1. **개발 방법**
* **File 관련 system call들은 Create, remove, open, close, filesize, read, write, seek, tell등이 있으며 pintos manual을 참고해 이를 userprog/syscall.c에 추가 할 것이다.**
* **함수들을 구현하기 위해 thread 구조체에 열린 file pointer를 저장하는 struct file \*f[130] 변수와 현재 file descriptor를 지칭하는 int fd 변수를 추가 할 것이다.**
* **lock\_acquire, lock\_release 함수를 통해 critical section을 설정할 것이다.**
* **sema-up , sema-down 함수를 통해 semaphore 방식의 synchronization이 이루어 질 수 있도록 할 것이다.**
* **wait 함수를 위해 user\_defined 함수인 pass\_status , is\_child\_exit , is\_wait\_called, wait\_sema\_down, wait\_sema\_up 함수를 구현할 예정이며 그 기능은 다음과 같다.**

**1) pass\_status**

**Process가 종료되었을 때, 그 process의 exit status를 저장하거나 parameter로 주어진 tid의 exit status를 return하는 함수이다. syscall.c의 exit 함수에서는 exit status를 저장하기 위해 호출되고, process.c의 process\_wait 함수에서는 return 값으로 사용된다.**

**2) is\_child\_exit**

**각각의 thread의 child가 종료되었는지를 검사하는 함수이다. syscall.c의 exit 함수에서는exit 여부를 저장하기 위해 호출되고, process.c의 process\_wait 함수에서는 인자로 전달된 tid에 해당하는 thread가 exit되었는지 확인하기 위해 쓰인다.수에서 호출되며, 만약 child가 exit되었는데 process\_wait 함수 내에서 sema\_down을 하면 안되기 때문에 (exit에서 sema\_up을 하지 않기 때문에 교착상태에 빠지게 된다.) 이를 방지한다.**

**3) is\_wait\_called**

**Process\_wait 함수가 호출되었는지 검사하는 함수이다. Child process가 parent process의 wait 함수 호출 전에 종료될 때 sema\_down이 안 되었는데도 sema\_up을 하는 것을 방지하기 위해 사용된다. Process\_wait 함수에서는 wait 호출 여부를 저장하기 위해 호출되고, exit에서는 wait의 호출 여부를 검사하기 위해 호출된다.**

**4) wait\_sema\_down**

**Sema\_down과 같은 기능을 하지만, is\_wait\_called 함수를 통한 wait 호출 여부를 저장하는 변수의 수정을 atomic하게 하기 위해 정의한 함수이다. 만약 wait 호출 여부를 저장하는 변수의 수정이 atomic하게 되지 않으면 다음과 같은 문제가 발생할 수 있다.**

**“Process\_wait 함수에서 wait 호출 여부를 저장하는 변수에 true 저장 -> preemption으로 child process 실행 -> child가 exit 함수를 호출 -> sema\_down이 되지 않았음에도 sema\_up 실행 -> kernel panic”**

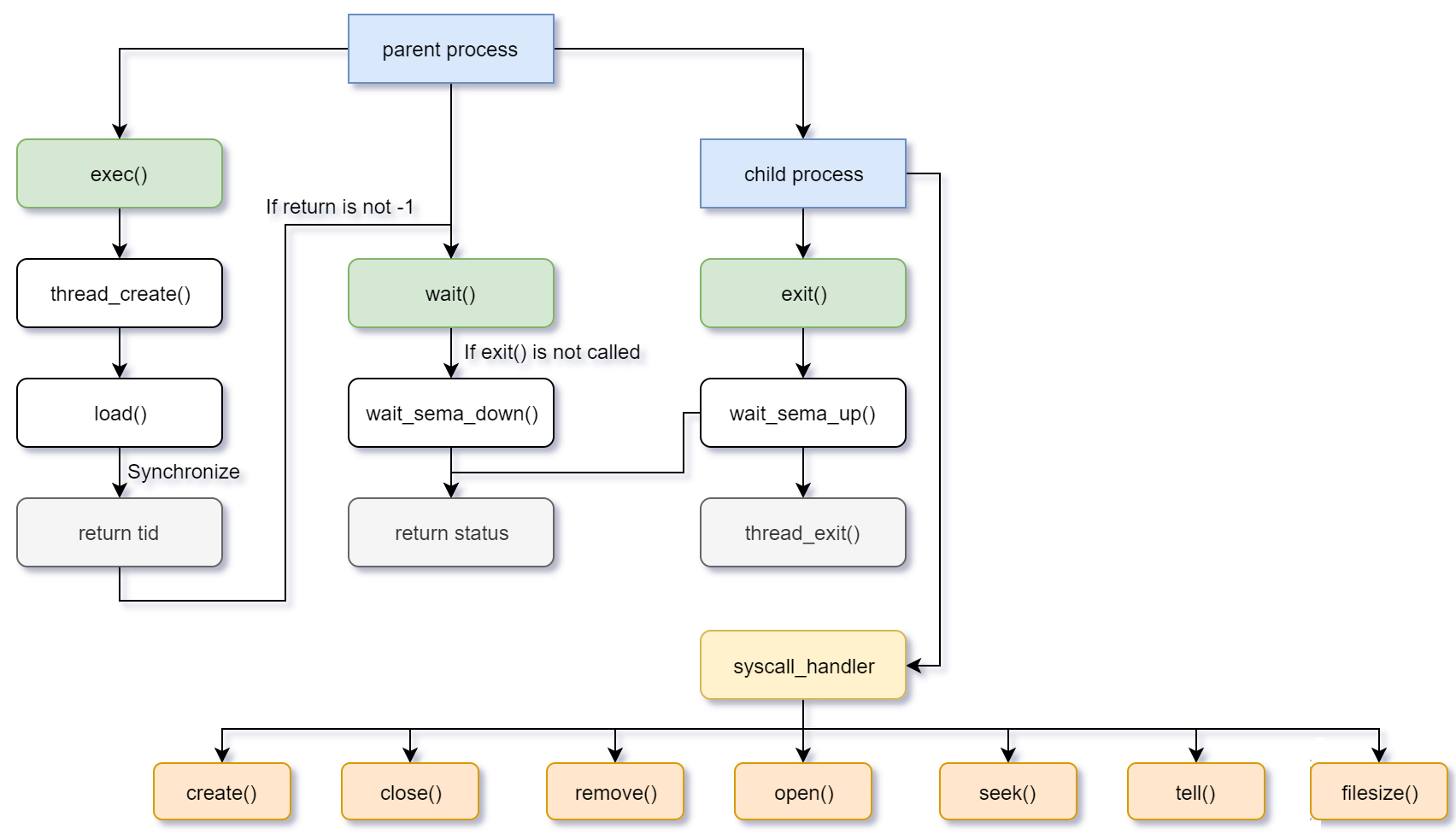
**위와 같은 문제 상황을 해결하기 위해 정의되었다.**

**5) wait\_sema\_up**

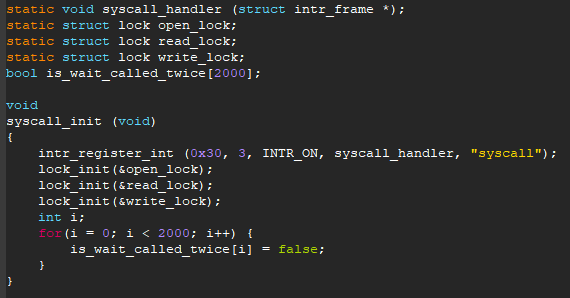
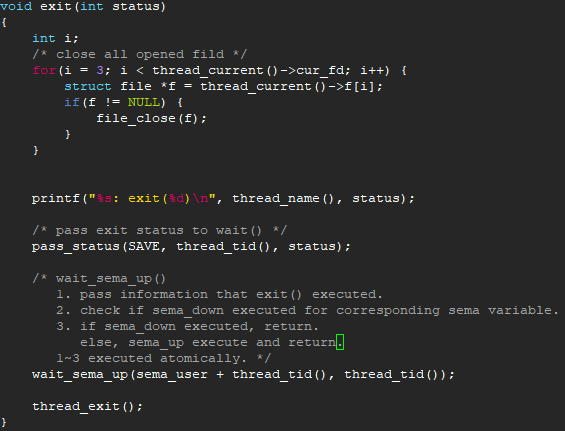
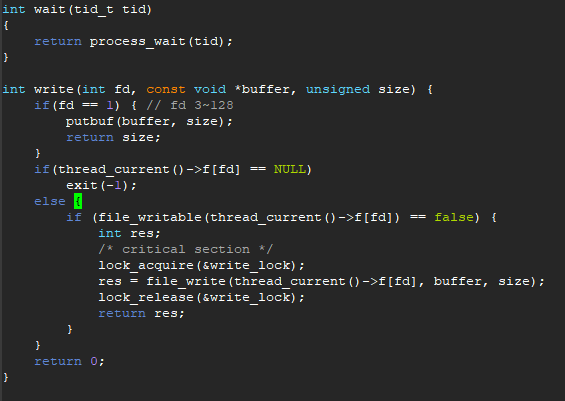
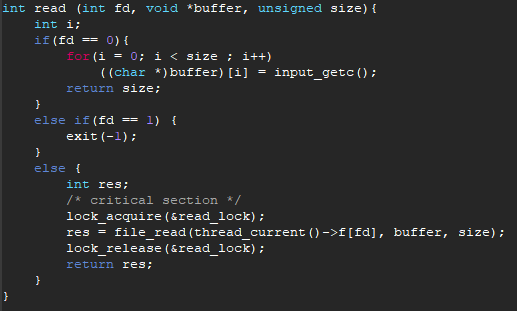
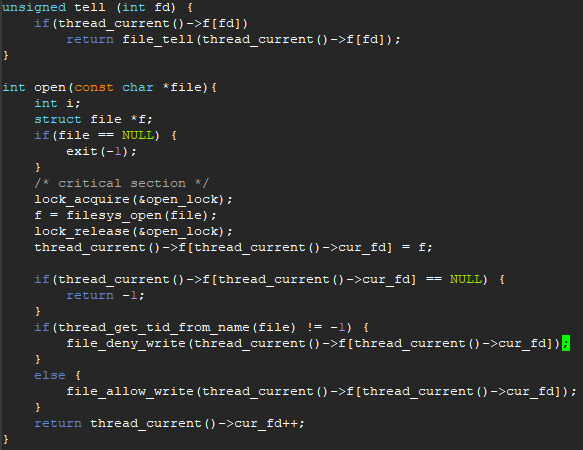
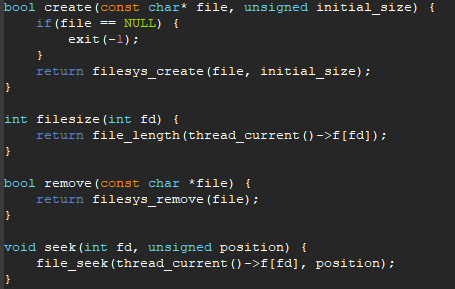
**Sema\_up과 동일한 기능을 하는 함수로, is\_child\_exit 함수를 통한 변수의 수정을 atomic하게 하기 위해 저장되었다. 또한 is\_wait\_called 함수를 통해 wait 함수에서 sema\_down이 실행되었는지 검사하고, 만약 실행되지 않았다면 sema\_up을 실행하지 않는다.**

* 1. **연구원 역할 분담**
* **장지훈 : pintos manual 분석 및 synchronize ,syscall, rox, wait 구현**
* **안시현 : pintos manual 분석 및 bad 관련 test, 보고서 작성**

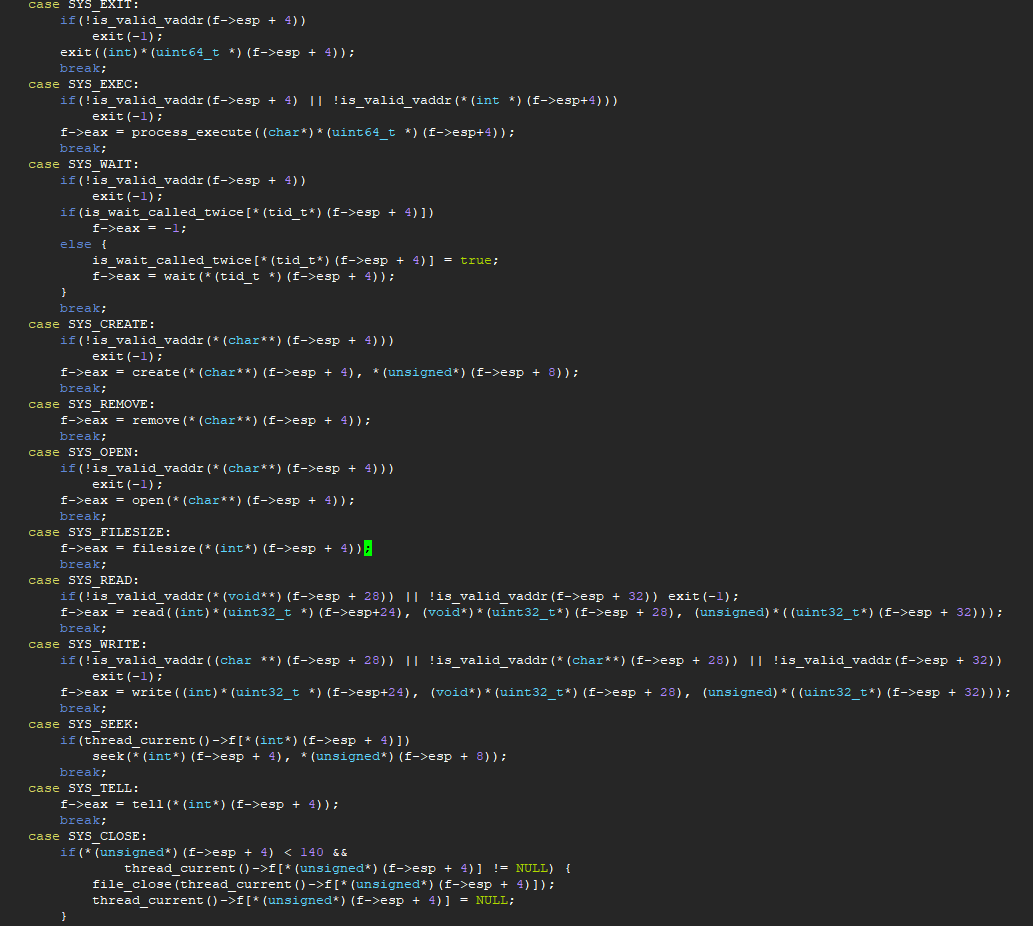
1. **연구 결과**
   1. **합성 내용**

* **Flow chart**
* 
  1. **제작 내용**
* **Create : 새로운 파일을 생성한다. 기존에 정의되어 있는 filesys\_create를 이용한다. 실패 시 false를 반환한다.**
* **Remove : File Remove 함수로 기존 파일을 삭제한다. 기존에 정의되어 있는 filesys\_remove 함수를 이용하며 실패 시 false를 반환한다.**
* **Open : File open 함수로서 전달된 filename을 가지는 file을 열어준다. Filesys\_open을 통해 file을 open하고 이를 현재 thread의 f배열에 추가 한다. 실패시 -1을 반환한다. open하는 동안은 다른 thread의 접근 방지가 보장되어야 하기 때문에 critical section으로 설정하였다. 또한 pintos는 실행중인 프로그램의 executable file이 지워지지 않기를 원하기 때문에 file\_deny\_write와 file\_allow\_write를 이용하여 이를 구현 하였다.**
* **Close : File을 Close 해주는 함수로서 file\_close 함수를 이용한다.**
* **Read : file을 읽어주는 함수로서 file\_read함수를 이용하였다. File\_read가 수행되는 동안에는 critical section으로 지정함으로써 다른 thread의 접근을 방지하였다.**
* **Write : file에 write하는 함수로서 file\_write 함수를 이용하였다. File\_write가 수행되는 동안에는 critical section으로 지정함으로써 다른 thread의 접근을 방지하였다. 또한 f->deny\_write를 return해주는 file\_writable 함수를 정의하여 write 가능한 상태에만 wirte를 진행하도록 하였다.**
* **Seek : file의 position 값을 바꾸어주는 함수로서 file\_seek 함수를 이용하여 구현하였다.**
* **Tell : file의 position 값을 찾아주는 함수로서 file\_tell 함수를 이용하여 구현하였다.**
* **Filesize : file size를 구해 주는 함수로서 file\_length 함수를 이용하여 구현하였다.**

**Syscall 관련 코드 첨부**

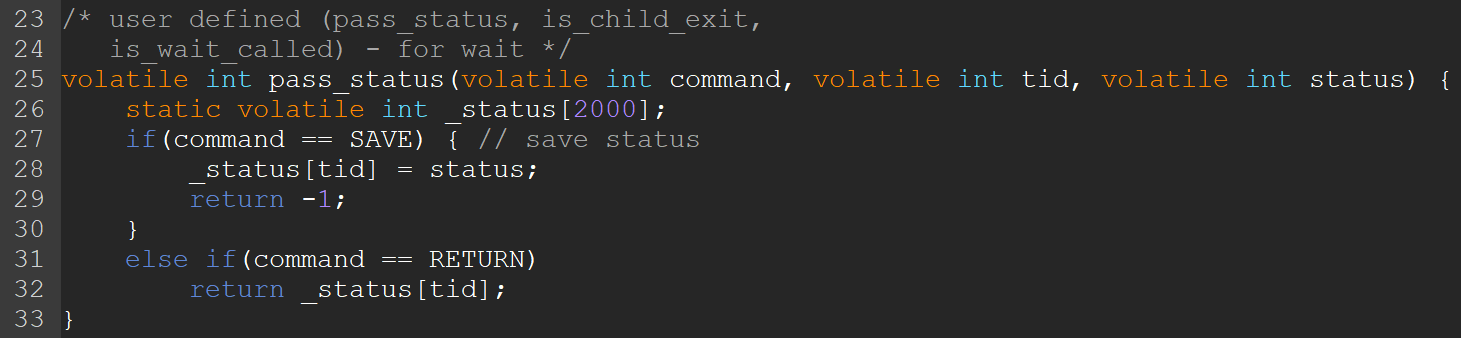
**Syscall\_handler 부분 코드 첨부이다.**



**wait를 구현하기 위해 pass\_status, is\_child\_exit, is\_wait\_called, wait\_sema\_down, wait\_sema\_up의 5개 함수를 process.c에 정의했다.**

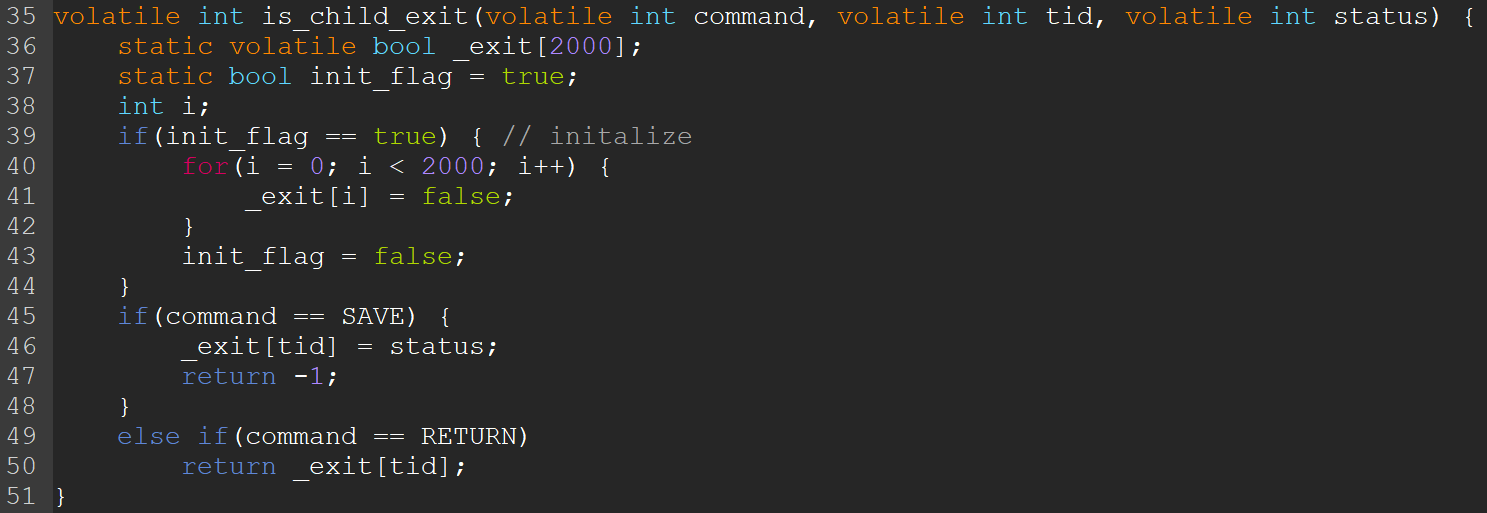
**1) pass\_status**

**Process가 종료되었을 때, 그 process의 exit status를 저장하거나 parameter로 주어진 tid의 exit status를 return하는 함수이다. syscall.c의 exit 함수에서는 exit status를 저장하기 위해 호출되고, process.c의 process\_wait 함수에서는 return 값으로 사용된다. Command가 SAVE일 때는 값을 저장하고, RETURN일 때는 저장된 값을 return한다.**



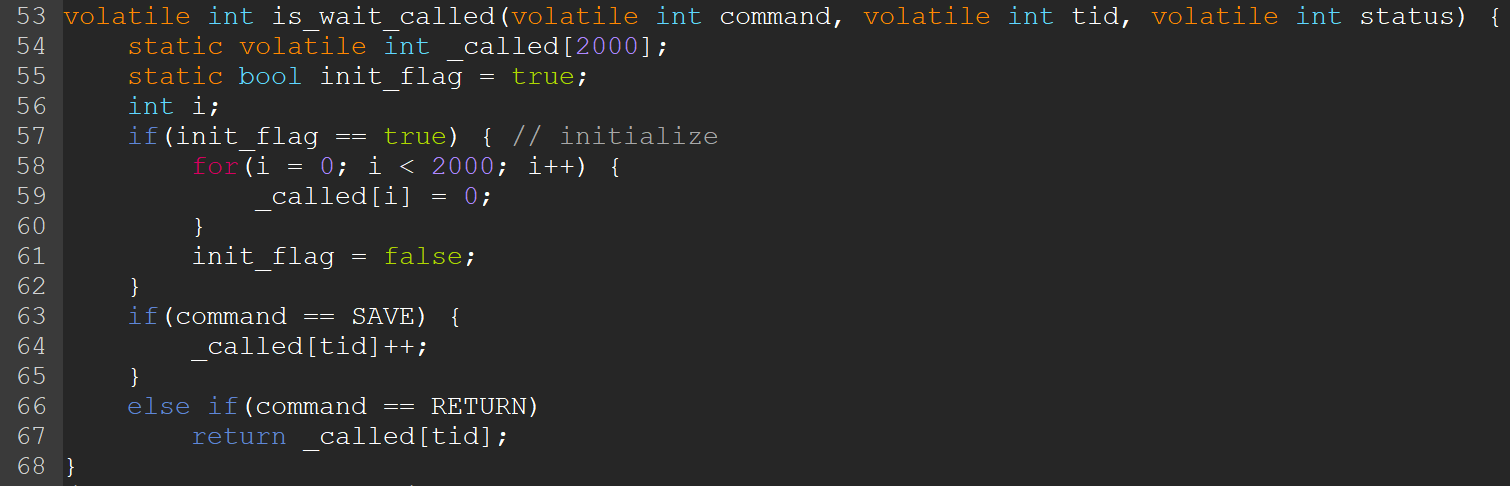
**2) is\_child\_exit**

**각각의 thread의 child가 종료되었는지를 검사하는 함수이다. syscall.c의 exit 함수에서는 exit 여부를 저장하기 위해 호출되고, process.c의 process\_wait 함수에서는 인자로 전달된 tid에 해당하는 thread가 exit되었는지 확인하기 위해 쓰인다.**



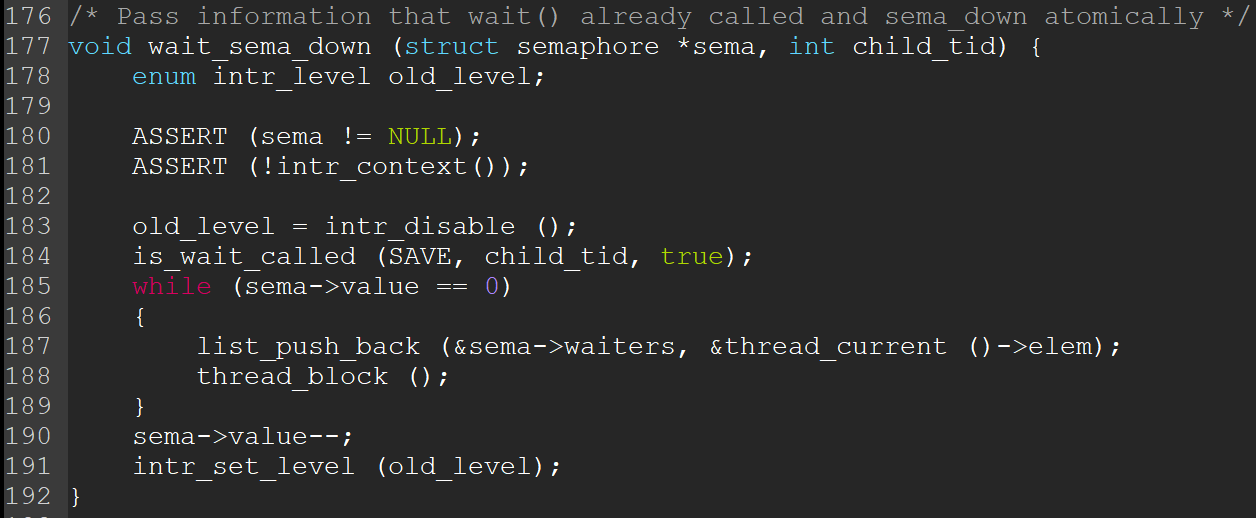
**3) is\_wait\_called**

**Process\_wait 함수가 호출되었는지 검사하는 함수이다. Child process가 parent process의 wait 함수 호출 전에 종료될 때 sema\_down 호출 전 sema\_up을 하는 것을 방지하기 위해 사용된다. Process\_wait 함수에서는 wait 호출 여부를 저장하기 위해 호출되고, exit에서는 wait의 호출 여부를 검사하기 위해 호출된다.**



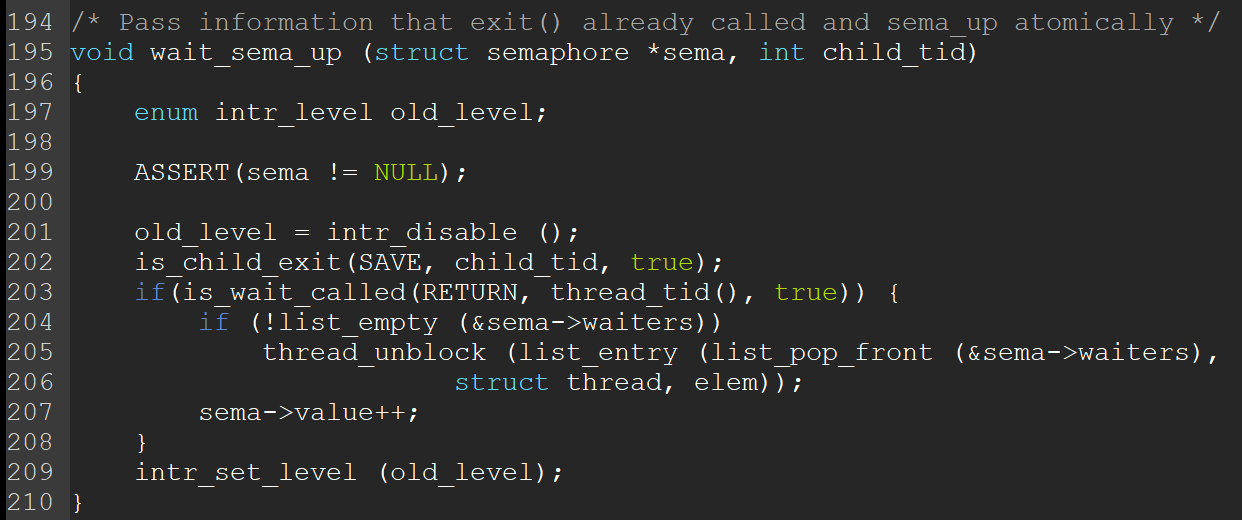
**4) wait\_sema\_down**

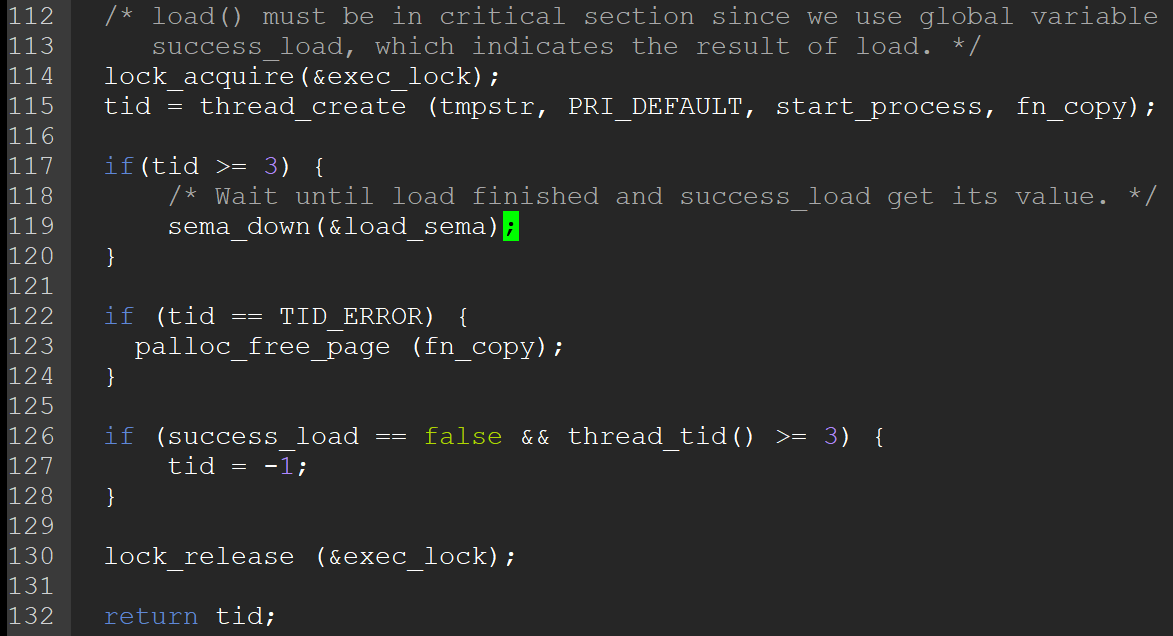
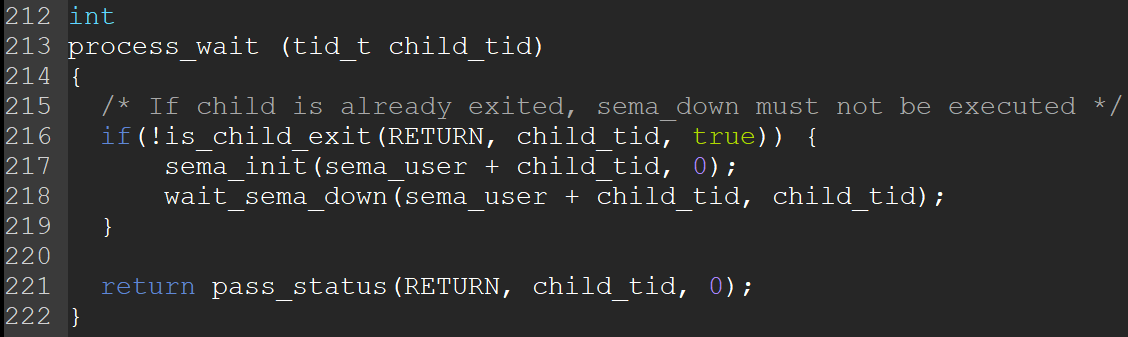
**Sema\_down과 같은 기능을 하지만, is\_wait\_called 함수를 통한 wait 호출 여부를 저장하는 변수의 수정을 atomic하게 하기 위해 정의한 함수이다. 코드는 다음과 같다.**



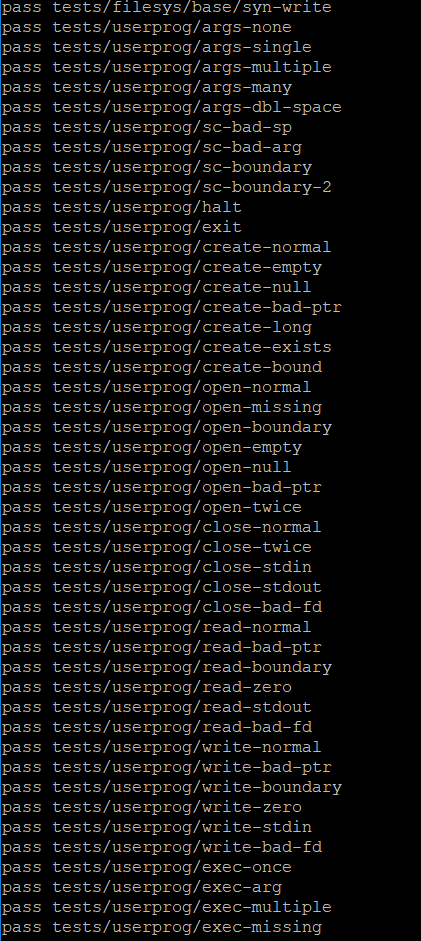
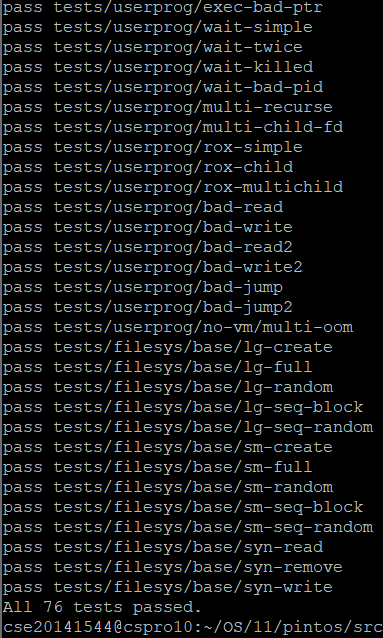
**5) wait\_sema\_up**

**Sema\_up과 동일한 기능을 하는 함수로, is\_child\_exit 함수를 통한 변수의 수정을 atomic하게 하기 위해 저장되었다. 또한 is\_wait\_called 함수를 통해 wait 함수에서 sema\_down이 실행되었는지 검사하고, 만약 실행되지 않았다면 sema\_up을 실행하지 않는다.**

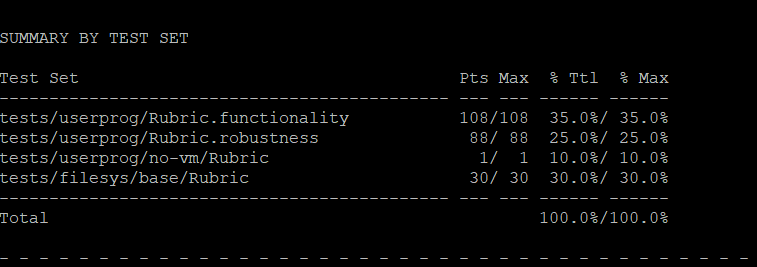


* **Process\_execute : thread\_create 함수가 호출된 후 process가 load될 때, 이를 기다리기 위해 0으로 초기화된 sema\_down과 sema\_up을 사용한다. 또한 load를 임계구역으로 설정하여 한 process의 load 성공 여부를 확실하게 알기 전까지 다른 process가 load를 실행하지 못하도록 구현하였다. 다음은 process\_execute에서 semaphore와 lock이 사용되는 부분과, load에서 sema\_up을 하는 부분이다.**
* 
* 
* **Process\_wait : 0으로 초기화된 semaphore로 child의 exit을 기다린다. 코드는 다음과 같다.**
* 
  1. **시험 및 평가 내용**

**Make check 결과 첨부**

* 
* 

**Make\_grade.jpg**



1. **기타**
   1. **연구 조원 기여도**

* **안시현(50%)**
* **장지훈(50%)**
  1. **소감**
* **안시현 : 이번 pintos project를 통해 file system 관련 system call을 구현하면서 운영체제에서 파일 관련된 기능들이 어떻게 작동되는지에 관해 더 자세히 이해할 수 있었다. 또한 , Synchronization을 구현하면서 그동안 programming 언어나 다른 수업에서 배웠던 semaphore 방식에 대해 조금 더 이해 할 수 있게 되었고, 실제로 동기화를 구현함으로써 thread나 read, write 사이에서의 동기화의 원리를 이해할 수 있었다.**
* **장지훈 : 수업으로 이론만 습득하였던 critical section의 보호, semaphore 등을 직접 사용하고 코드로 구현해보면서 OS 설계에서 고려해야 할 점, 다중 thread 프로그램을 작성할 때 주의해야 할 점에 대해 생각해 볼 수 있는 좋은 기회였다. 또한 여러 차례, 여러 종류의 debugging을 하면서 engineer로서의 설계 능력에 밑거름으로 삼을 수 있었다.**