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

**(설계 프로젝트 수행 결과)**

과목 명 :[CSE4070] 운영체제

담당 교수 :김영재 교수님

조 / 조원 : 58조 / 20151601(장진영), 20151600(장승민)

개발 기간 :2019.11.04 ~ 2019.11.17

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

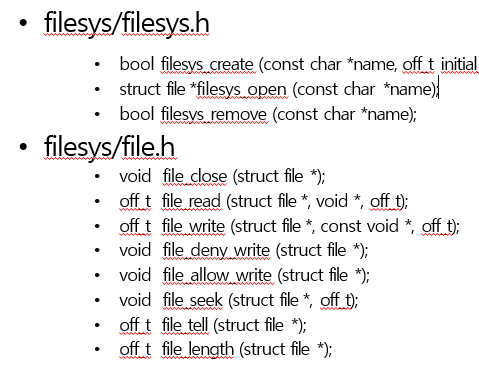
**제출일 : 2019.11.17**

**참여 조원 : 장승민 장진영**

1. **개발 목표**

* pintos에서 system call을 구현하여 user program을 모두 실행시키는 것이 최종 목표이다. Project 1(User Program Basic)에서 기본적인 System Call 과 argument passing을 구현하였고, 이번 Project 2에서는 파일 시스템에 관련된 System Call을 구현하고 특정 Thread가 일정 부분을 반복하여 수행하고 다르 Thread의 수행을 대기하는 상태인 동기화기법인 busy waiting에서 효율적인 부분을 개선한 semaphore 방법으로 변경하여 사용한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
2. **Base File System**
   * + - 1. **Pintos에서 요구하는 파일 시스템 관련 System call을 구현하기 위해서는 Pintos에서의 기본적인 파일 시스템 구조를 이해해야한다. 명세서에서 주어진 아래의 API들을 활용하여 구현한다.**

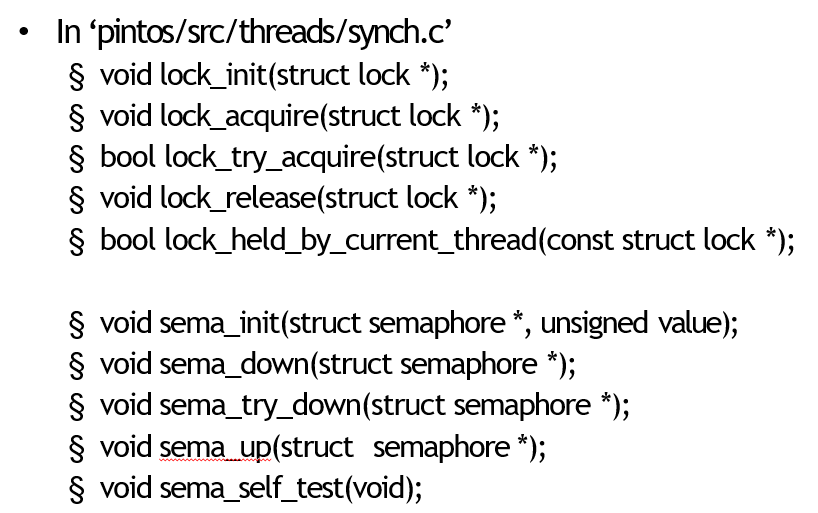


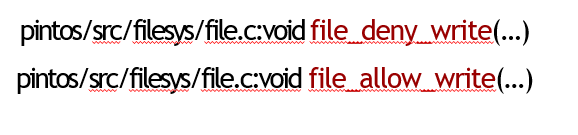
* + - * **int read ( int fd, void \*buffer, unsigned size )   
        project1에서는 fd가 0인 경우 keyboard로 부터 입력을 받는 것만 읽어서 buffer에 넣었다면 project2에서는 thread의 file안에 있는 내용들도 읽어서 buffer에 넣는 것을 추가해야 한다. 이를 위해 파일 디스크립터를 이용하여 파일 객체를 검색하고, 파일 객체를 활용하여 buffer에 넣는다.**
      * **int write ( int fd, const void \*buffer, unsigned size )   
        project1에서는 fd가 1인 경우 buffer에서 읽어서 바로 console창에 출력을 했다면 project2에서는 buffer에 있는 내용을 thread의 file안에 write하는 것을 추가해줘야 한다. 이를 위해서는 thread안에 struct file type의 file배열을 설정하고 초기화해야 한다.**
      * **int create ( const char \*file, unsigned initial\_siez )  
        file이름에 해당하는 file을 create해야 한다.**
      * **int remove ( construct char \*file )  
        file이름에 해당하는 file을 remove해야 한다.**
      * **int open ( const char \*file )  
        file이름에 해당하는 file을 open한 후 현재 thread에 file에 넣어줘야 한다.**
      * **int filesize ( int fd )  
        현재 thread의 fd에 해당하는 file의 size(길이)를 구한다.**
      * **void seek ( int fd, unsigned position )  
        thread에 저장되어 있는 position을 재설정 한다.**
      * **unsigned tell ( int fd )**

**thread에 저장되어 있는 position을 구한다.**

* + - * **void close ( int fd )**

**현재 thread의 fd에 해당하는 file을 close한다.**

* + 1. **Synchronization(Protect Critical Section)**
       - **Multithreading 환경에서의 서로 다른 Thread에서 Critical Section을 동시에 접근하면 문제가 생기므로 적절한 Synchronization이 필요하다. 명세서에서 주어진 아래의 API들을 적절하게 활용하여 구현한다.**
       - ****
       - **parent thread가 child thread가 생겼을 때 child thread가 끝날때까지 parent thread를 block하여 orphan process생성하여 program error가 뜨는 것을 방지하기 위해 semaphore를 사용하였다. 또한 semaphore로 인해서 chile thread가 접근하는 영역에 parent thread가 접근할 수 없으므로 parent thread와 child thread사이에 synchronization도 보장할 수 있었다.**
       - **semaphore를 thread 가 load되는 것을 기다리고 thread가 wait하는 것을 기다리고 thread를 exit하기 전에 status를 return하는 것을 보장하기 위해 사용된다. 그러므로 load 함수 exec함수 wait함수 exit함수에서 semaphore를 사용해주어야 한다.**
    2. **Denying Writes to Executables**
       - **프로세스는 메모리에 로드를 완료한 후에 execution되므로 실제 exectuable file은 지워져도 문제가 없다. 하지만 Pintos에서는 실행중인 프로그램의 executable file이 지워지지 않기를 바란다. 이를 구현하는 것이다. 명세서에서 2가지 API를 제시하고 이를 이용하여 구현한다고 제시해주었다.**



* + - * **한번 메모리에 load한 후에 수정을 막아야 하고 process가 종료되고 나서 수정을 다시 허용해줘야 하도록 load함수와 thread\_exit함수를 수정해야 한다.**
  1. **개발 내용**
     1. **Base File System**
        + - **Base File System을 구현하기 위하여 syscall handler에 각각의 base file system에 해당하는 system call을 추가하였고 system call에 해당하는 base file system을 진행하기 전에 syscall handler에 parameter로 넘어온 interrupt에 해당하는 intr\_frame \*f의 esp에 들어있는 값들이 kernel virtual address에 해당하면 안되기 때문에 is\_user\_vaddr을 통해서 확인하였다.**
          - **int read ( int fd, void \*buffer, unsigned size )   
            file\_read함수를 사용하여 file안의 내용을 size만큼 읽어서 buffer에 저장하고 buffer에 저장한 글자 수를 return한다.**
          - **int write ( int fd, const void \*buffer, unsigned size )   
            file\_write함수를 사용하여 buffer의 내용을 size만큼 읽어서 file에 쓰고 쓰여진 글자 수를 return한다.**
          - **int create ( const char \*file, unsigned initial\_size )  
            file을 name으로 하고 크기가 initial\_size인 file을 생성하기 위해 filesys\_create함수를 사용한다.**
          - **int remove ( construct char \*file )  
            이름이 file인 것을 지워주기 위해 filesys\_remove함수를 사용한다.**
          - **int open ( const char \*file )  
            지금 syscall을 실행하고 있는 thread의 structure에 있는 file에 parameter로 넘어온 file name에 해당하는 file을 filesys\_open함수를 사용하여 open하고 file descriptor 3부터 시작해서 비어있는 곳에 저장한다. file descriptor 0은 read이고 file descriptor 1은 write이므로 3부터 시작한다.**
          - **int filesize ( int fd )  
            현재 thread의 fd에 해당하는 file의 사이즈를 file\_length함수를 이용하여 구한다.**
          - **void seek ( int fd, unsigned position )**

**현재 thread의 fd에 해당하는 file의 position을 file\_seek함수를 이용하여 저장한다.**

* + - * + **unsigned tell ( int fd )**

**현재 thread의 fd에 해당하는 position을 file\_tell함수를 이용하여 구한다.**

* + - * + **void close ( int fd )**

**현재 thread의 fd에 해당하는 file을 file\_close함수를 이용하여 닫는다**

**2.Protect Critical Section(Synchronization)**

* + - * **pid\_t exec(const char \*cmd\_line)  
        child thread에 대한 thread가 load가 완료될 떄까지 sema\_down(load\_sema)을 통해 기다려야 한다. Process execute를 리턴한다. 그 후에 process\_execute에 있는 sema\_down을 통해 기다린다.**
      * **Static void start\_process(void \*file\_name\_)  
        child thread에 대한 load가 완료되면 sema\_up(load\_sema)을 통해 exec함수에서 기다리고 있는 thread에 알려줘야 한다.**
      * **int process\_wait(tid\_t child\_tid)  
        parent thread는 child thread가 exit될 떄까지 sema\_down을 하여 기다린다. child thread의 process가 exit이 되면 해당 thread에 해당하는 child\_elem을 parent thread가 갖고 있는 child들의 child\_elem list에서 지운 후 sema\_up을 통해 child thread의 thread를 지워준다.**
      * **void process\_exit(void)**

**process\_wait에서 sema\_down을 통해 parent thread가 기다리고 있고 child thread가 진행돠는데 진행이 끝나서 process\_exit에 들어오면 sema\_up을 해줘서 기다리고 있는 parent\_thread에 알려준다. 알려준 후에는 child\_elem을 지운후 남아있는 child thread를 말끔히 지워준다.**

**3. Denying Writes to Executables**

* + - * **file\_deny\_write(struct \*file file)**

**parameter file을 접근할 수 없도록 deny한다.**

* + - * **file\_allow\_write(struct\* file file)**

**parameter file의 접근 deny를 풀어주는 역할을 한다.**

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

**11.04 ~ 11.07 : Base File System 이해 및 API 사용 방법 이해**

**11.07~ 11.08 : read, write, create, remove, open, filesize, seek, tell, close 구현(동기화 적용 X)**

**11.08 ~ 11.11 : Denying Writes to Execitables 구현**

**11.11 ~ 11.14 : Synchronization 구현**

**11.14~ 11.16 : multi-oom(test case) 구현**

* 1. **개발 방법**
     1. **Base File System**
        1. **pintos/src/lib/syscall-nr.h에 존재하는 각 system call에 해당하는 번호를 바탕으로 pintos/src/userprog/syscall의 syscall\_handler(struct intr\_frame \*f)에 f->esp가 해당 system call을 저장하고 있을 때를 추가하였다.**
        2. **int read ( int fd, void \*buffer, unsigned size )   
           pintos/src/userprog/syscall.c의 read(int fd, void \*buffer, unsigned size)에 파일에 동시 접근이 일어날 수 있으므로 LOCK을 이용하였다. 파일 디스크립터를 이용하여 파일 객체를 검색하고, 파일 디스크립터가 0일 경우 키보드에 입력을 버퍼에 저장 후 버퍼의 저장한 크기를 리턴한다. 이 때, input\_getc()함수를 이용한다. 또한, 파일 디스크립터가 0이 아닐 경우 파일의 데이터를 크기 만큼 저장 후 읽은 바이트 수를 리턴한다.**
        3. **int write ( int fd, const void \*buffer, unsigned size )   
           pintos/src/userprog/syscall.c의 wrtie(int fd, const void \*buffer, unsigned size)에 파일에 동시 접근이 일어날 수 있으므로 LOCK을 이용하였다. 파일 디스크립터를 이용하여 파일 객체를 검색하고, 파일 디스크립터가 1일 경우 버퍼에 저장된 값을 화면에 출력 후 버퍼의 크기를 리턴한다. 이 때, putbuf()함수를 이용한다. 파일 디스크립터가 1이 아닐 경우 버퍼에 저장된 데이터를 크기 만큼 파일에 기록후 기록한 바이트 수를 리턴한다.**
        4. **int create ( const char \*file, unsigned initial\_size )  
           pintos/src/userprog/syscall.c의 create( const char \*file, unsigned initial\_size) 에 file을 open하기 위해 filesys\_create(file,initial\_size) 함수를 통해 리턴한다. 이 함수는 파일 이름과 파일 사이즈를 인자 값으로 받아 파일을 생성하는 함수이다.**
        5. **int remove ( construct char \*file )  
           pintos/src/userprog/syscall.c의 remove ( construct char \*file )에 file을 remove하기 위해 filesys\_remove(file) 함수를 통해 리턴한다. 이 함수는 파일 이름에 해당하는 파일을 제거하는 함수이다.**
        6. **int open ( const char \*file )  
           pintos/src/userprog/syscall.c의 open ( const char \*file )에 file을 open하기 위해 filesys\_open(file)을 추가하였고 , 해당 파일 객체에 파일 디스크립터를 부여하였다. 파일 디스크럽터를 리턴하고, 해당 파일이 존재하지 않으면 -1을 리턴합니다.**
        7. **int filesize ( int fd )**

**pintos/src/userprog/syscall.c의 filesize ( int fd )에 파일 디스크립터를 이용하여 파일 객체를 검색한다. file의 size를 구하기 위해 file\_length함수를 사용한다. File\_length함수는 파일 크기를 알려주는 함수이다. 해당 파일이 존재하지 않으면 -1을 리턴한다.**

* + - 1. **void seek ( int fd, unsigned position )**

**pintos/src/userprog/syscall.c의 seek ( int fd, unsigned position )에 파일 디스크립터를 이용하여 파일 객체를 검색한다. file의 position을 parameter로 넘어온 position으로 설정하기 위해 file\_seek(fptr(fd), position)을 추가하였다. 해당 열린 파일의 위치(Offset)를 position만큼 이동한다.**

* + - 1. **unsigned tell ( int fd )**

**pintos/src/userprog/syscall의 tell ( int fd )에 파일 디스크립터를 이용하여파일 객체를 검색한다. file에 저장되어 있는 position을 구하기 위해 file\_tell(fptr[fd])을 추가하였다. 해당 열린 파일의 위치를 반환한다.**

* + - 1. **void close ( int fd )**

**pintos/src/userprog/syscall.c의 close ( int fd )에 해당 파일 디스크립터에 해당하는 파일을 닫는다. . fd번째 file을 close하기 위해 file\_close(fptr(fd))을 추가하였다. 마지막으로, fptr(fd)를 NULL로 초기화 해준다.**

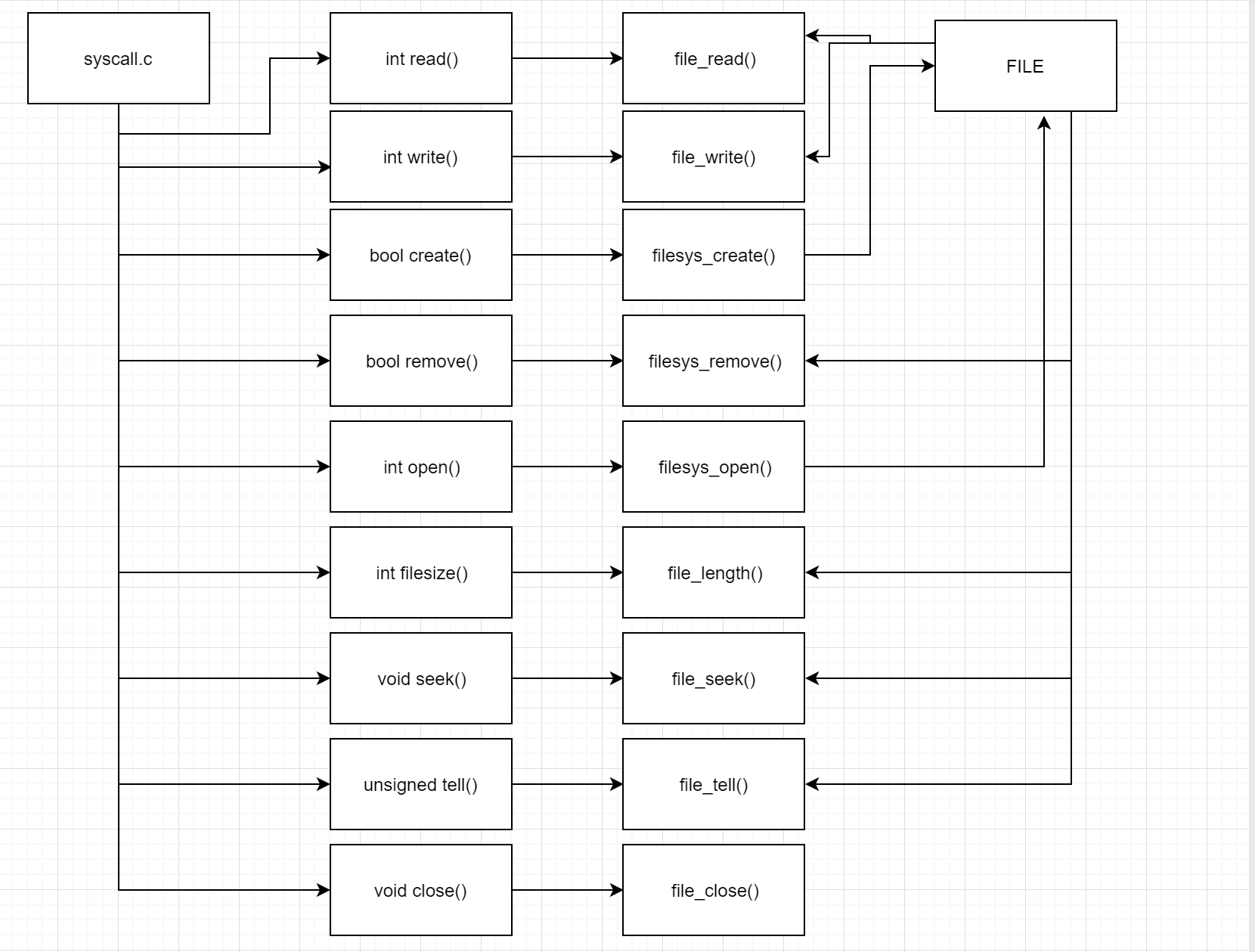
* + 1. **Protect Critical Section(Synchronization)**
       - * **pid\_t exec(const char \*cmd\_line)  
           pintos/src/userprog/syscall.c의 exec(const char \*cmd\_line)에 child thread에 대한 thread가 load가 완료될 떄까지 sema\_down(load\_sema)을 통해 기다려야 한다. Process execute를 리턴한다. 그 후에 process\_execute에 있는 sema\_down을 통해 기다린다.**
         * **Static void start\_process(void \*file\_name\_)  
           child thread에 대한 load가 완료되면 sema\_up(load\_sema)을 통해 exec함수에서 기다리고 있는 thread에 알려줘야 한다.**
       - **int process\_wait(tid\_t child\_tid)  
         pintos/src/userprog/process.c의 int process\_wait (tid\_t child\_tid)에 child thread가 완료될떄까지 기다리기 위해 sema\_down을 추가하였고 . child thread의 process가 exit이 되면 해당 thread에 해당하는 child\_elem을 parent thread가 갖고 있는 child들의 child\_elem list에서 지운 후 sema\_up을 통해 child thread의 thread를 지워준다.**
         * **void process\_exit(void)**

**process\_wait에서 sema\_down을 통해 parent thread가 기다리고 있고 child thread가 진행돠는데 진행이 끝나서 process\_exit에 들어오면 sema\_up을 해줘서 기다리고 있는 parent\_thread에 알려준다. 알려준 후에는 sema\_down을 통해 child\_elem을 지운후 남아있는 child thread를 말끔히 지워준다.**

* + 1. **Denying Writes to Executables**
       1. **pintos/src/userprog/process의 load()에서 success가 true, 즉 load가 성공하였을 경우 file\_deny\_Write를 하고, file\_close(file)을 통해 file\_allow\_Write를 자동으로 호출하여 file\_deny\_write을 풀어주었다.**
  1. **연구원 역할 분담**
* **같은 장소에서 서로 개념을 이해하고 함께 소통하면서 진행하였다. 따로 명확한 역할을 나눈 것이 아니고 서로에게 도움을 주며 모르는 부분에 대해서 피드백하고 대화하면서 진행하였다.**

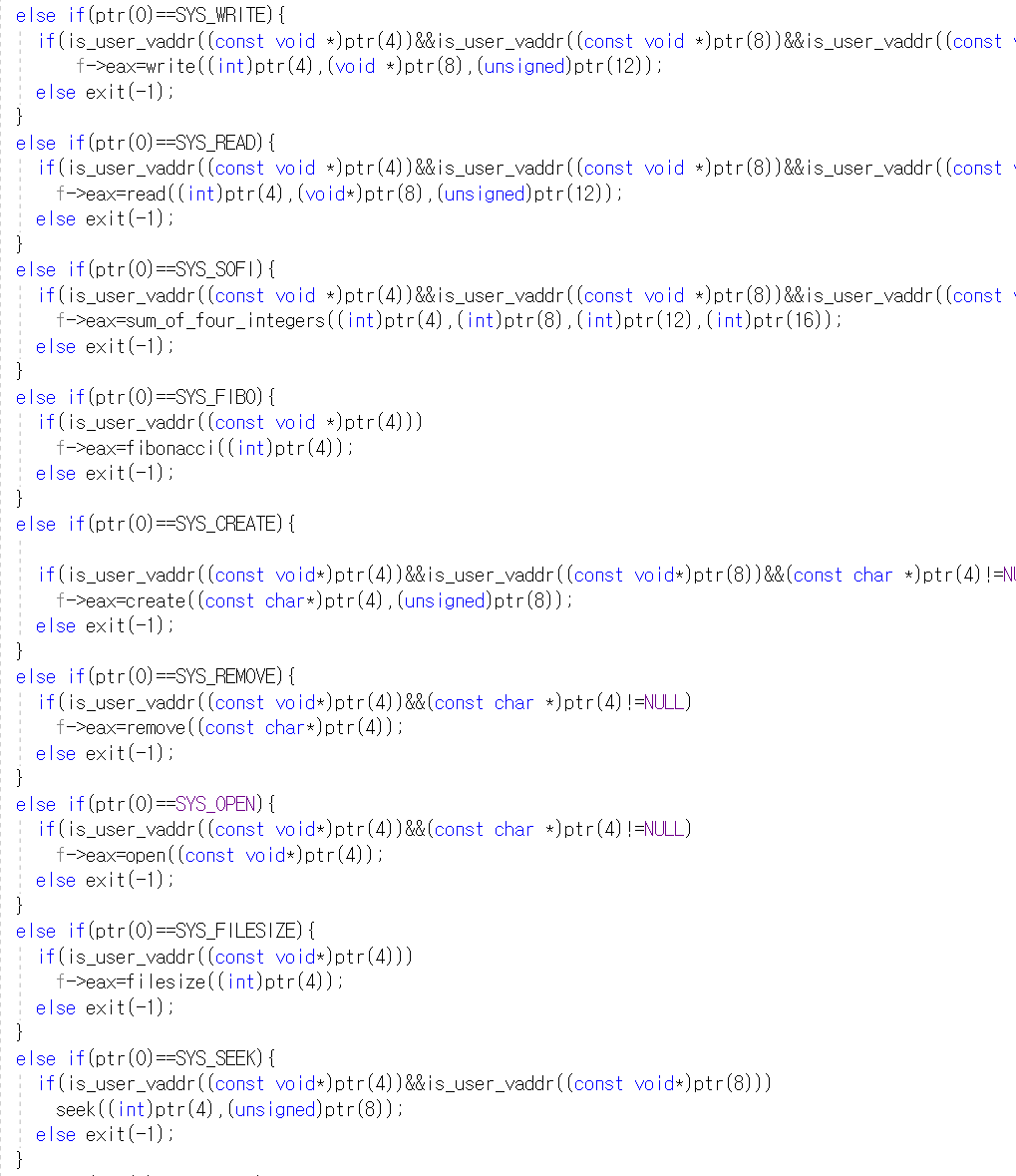
1. **연구 결과**
   1. **합성 내용**

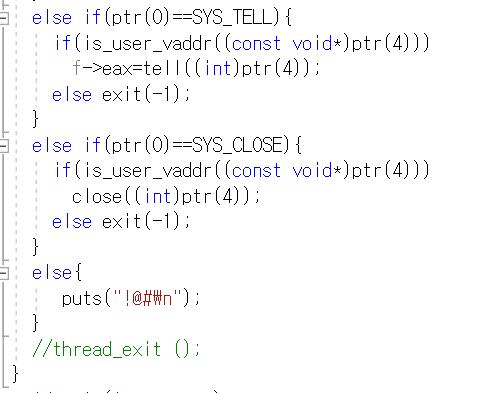
* **Flow Chart**

****

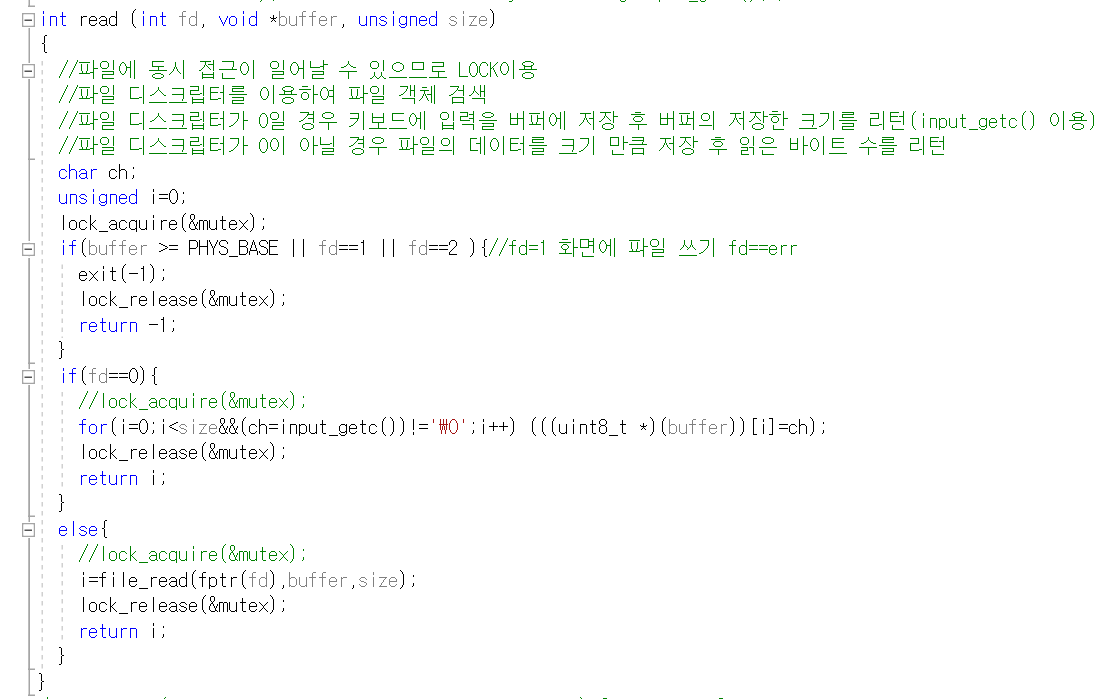
**B . 제작 내용**

* + 1. **Base File System**
* **static void syscall\_handler(struct intr\_fram \*f)**

****

****

**\* 함수 설명 : system call 에 해당하는 system call을 진행한다.  
\* 사용변수 :   
struct intr\_frame \*f : system call을 진행할 때 접근할 stack pointer를 갖고 있는 변수  
\* 내 용 : system call을 진행할 때 f에 저장된 stack pointer를 이용하여 memory에 접근하는데 memory에 접근하는 pointer들이 valid address인지 is\_user\_vaddr 함수를 통해 확인한다.**

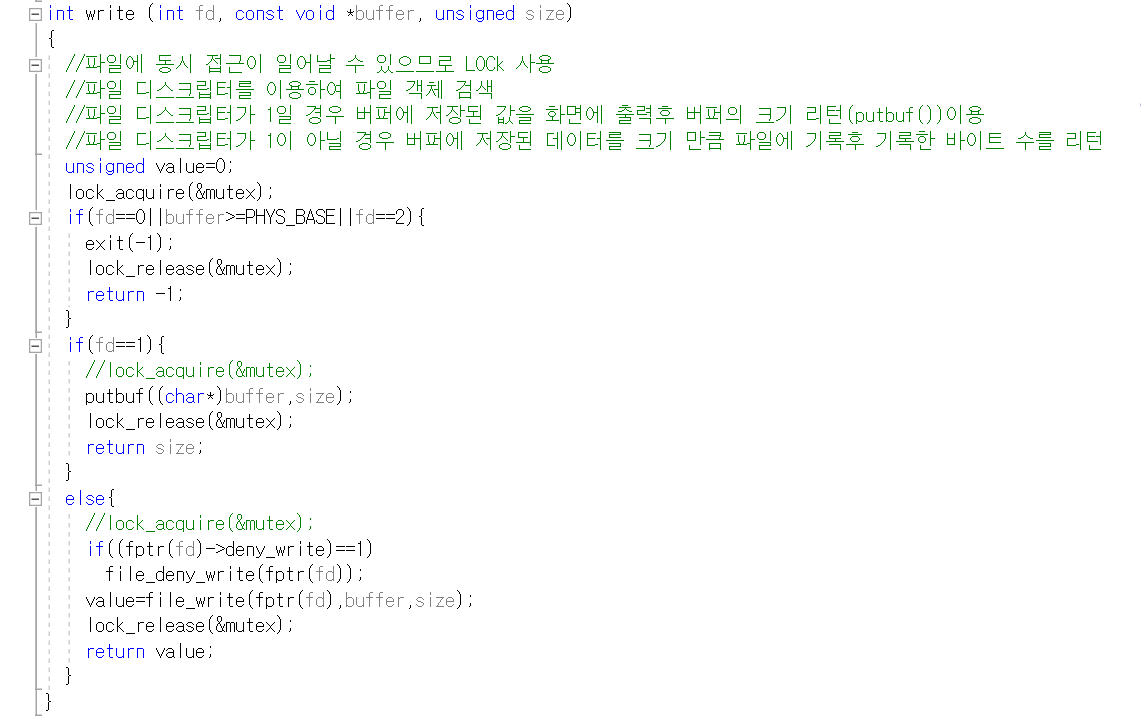
**- int read ( int fd, void \*buffer, unsigned size )   
**

**함수 설명 : keyboard나 file에 있는 내용을 읽어서 buffer에 저장하는 system call이다.  
\* 사용변수 :**

**int i : buffer에 저장한 글자의 수를 저장하는 변수이자 fd가 0일 때, for문의 변수가 되주는 변수.**

**\* 내 용 : 처음 시작할 때, lock\_acquire을 통해 lock을 걸어준다. fd가 0일때는 keyboard로 부터 input\_getc()함수를 통해서 입력을 받아서 buffer에 저장하고 저장한 글자 수를 return한다. fd가 0이 아닐 경우(fd가 1이거나 2이면)에는 fptr(fd)에 저장되어 있는 file이 비어있는지 확인하고 만약에 비어있다면 exit(-1)을 하고 비어있지 않다면 file\_read함수를 이용하여 fptr(fd)에 저장되어 있는 file의 내용을 buffer에 size만큼 저장하고 buffer에 올라간 글자수를 return 받아서 i에 저장한다. 모든 것이 다 끝나면, lock\_release함수를 통해 lock을 풀어준다.**

**- int write ( int fd, const void \*buffer, unsigned size )**

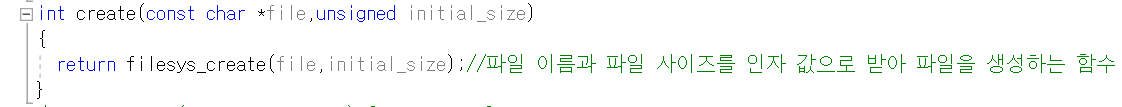
****

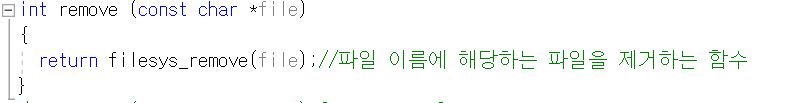
**\* 함수 설명 : buffer에 저장되어 있는 내용을 console이나 file에 write하는 system call이다.  
\* 사용변수 :  
unsigned value : buffer에서 읽어서 file에 write한 글자의 수를 저장하는 변수**

**\* 내 용 : 처음 시작할 때, lock을 걸어준다. fd = 1일 경우 putbuf함수를 이용하여 buffer에 저장되어 있는 size만큼의 글자를 console에 출력하고 출력한 글자 수를 return한다. Fd가 1이 아닐 경우, exit(-1)을 하고, buffer가 PHYS\_BASE보다 작고, fptr(fd)->deny\_write가 1과 같으면, file\_deny\_write를 해준다. Fptr(fd)가 deny\_write보다 크면, buffer에서 size만큼의 글자 수를 fptr(fd) 에 저장되어 있는 file에 적는다. 적은 글자수를 return 하여 value에 저장하고, lock을 풀어주고,**

**Return 한다.**

**- int create ( const char \*file, unsigned initial\_size )**

**  
\* 함수 설명 : file의 name과 initial size를 갖고 file을 생성하는 system call이다.  
\* 사용변수 : 없음  
\* 내 용 : filesys\_create함수를 이용하여 parameter로 넘어온 file과 initial\_size에 해당하는 file을 생성한다.**

**- int remove ( construct char \*file )**

**\* 함수 설명 : file의 name에 해당하는 file을 지우는 system call이다.  
\* 사용변수 : 없음  
\* 내 용 : filesys\_remove함수를 이용하여 parameter로 넘어온 file에 해당하는 file을 지운다.**

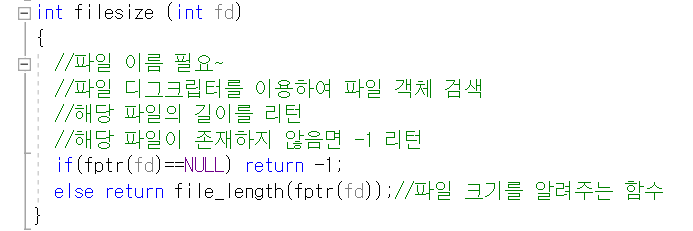
**- int open ( const char \*file )  
**

**\* 함수 설명 : file의 name에 해당하는 file을 여는 system call이다.  
\* 사용변수 :**

Struct file \*new : file에 해당하는 file을 open하여 저장하는 변수.

**int i : file descriptor를 설정해주는** 변수

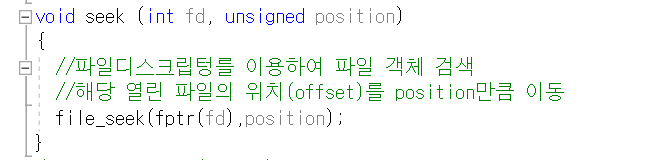
* + - * **\* 내 용 : 처음에, new 변수에 NULL로 할당하고, lock\_acquire함수를 통해 lock을 걸어준다. 저장하고 filesys\_open함수를 이용하여 제목이 file인 file을 open하여 new에 저장한다. new가 NULL일 경우는 -1을 return하고 아닐 경우 fptr(i)가 NULL인 것을 찾아서 strcmp를 통해 현재 thread의 name과 file이 복사가 된 리턴 값이 0이라면, file\_deny\_write를 해주고, fptr(i)에 new를 할당하고, lock\_release를 통해 풀어준다. 그 후에, i를 return해준다. 다 끝난 후에, lock을 풀어주고, -1을 리턴한다.**
* **int filesize ( int fd )**

****

**\* 함수 설명 : file descriptor가 fd에 해당하는 file의 길이를 구하는 system call이다.  
\* 사용변수 : 사용변수 없음.**

**\* 내 용 : fptr(fd)가 Null일 경우, -1을 리턴한다. 그렇지 않다면, file\_length 함수를 이용하여 file descriptor가 fd에 해당하는 file의 길이를 return한다.**

**- void seek ( int fd, unsigned position )**

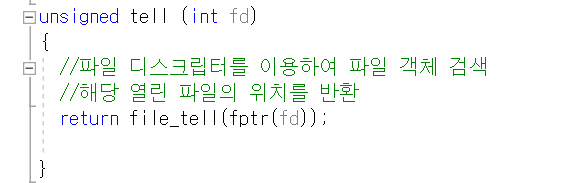
****

**\* 함수 설명 : file descriptor가 fd에 해당하는 file의 position을 parameter로 받아온 position으로 재설정하는 system call이다.**

**\* 사용변수 : 사용변수 없음.**

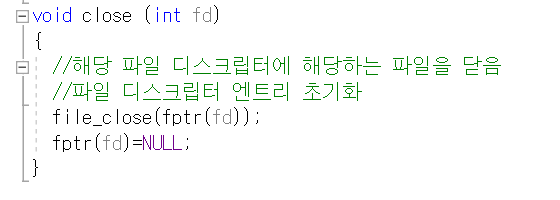
**\* 내 용 : file descriptor가 fd에 해당하는 file의 position을 parameter로 받아온 position으로 재설정 한다.**

**- unsigned tell ( int fd )**

**  
  
\* 함수 설명 : file descriptor가 fd에 해당하는 file의 position을 return하는 system call이다.  
\* 사용변수 : 사용변수 없음.**

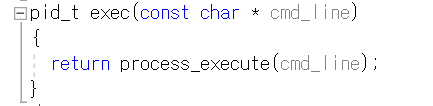
**\* 내 용 : file descriptor가 fd에 해당하는 file의 position을 return한다.**

**- void close ( int fd )**

**  
\* 함수 설명 : file descriptor가 fd에 해당하는 file을 close하는 system call이다.  
\* 사용변수 : 사용변수 없음.**

**\* 내 용 : file descriptor가 fd에 해당하는 file을 close 시키고 fptr(fd)에 NULL을 넣어서 초기화 시켜준다.**

* + 1. **Protect Critical Section(Synchronization)**

**- pid\_t exec(const char \*cmd\_line)  
**

**\* 함수 설명 : exec에 해당하는 system call이 들어왔을 때, process\_execute를 실행하여 리턴한다.   
\* 사용변수 : 사용변수 없음.**

**- tid\_t process\_execute(Const char \*file\_name)**

****

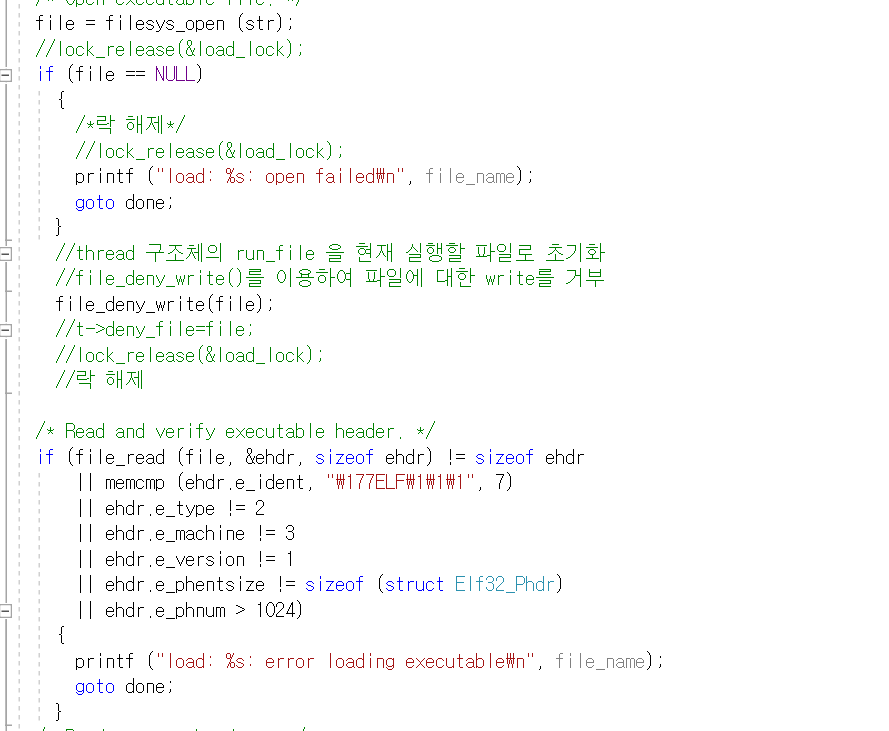
**\* 함수 설명 : exec에 해당하는 system call이 들어왔을 때 실행하는 base system이다.**

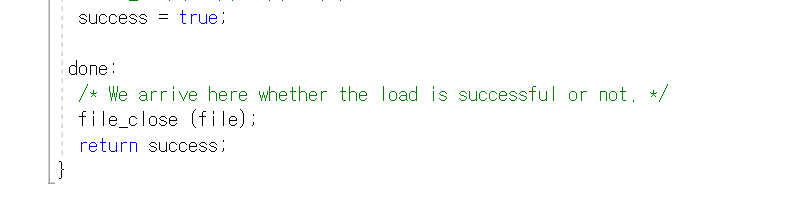
**\* 사용변수 :   
struct thread \*semathread : e에 해당하는 thread를 저장하는 변수**

**struct list\_elem \*e: children thread를 돌 때 사용되는 변수**

**\* 내 용 : process execute를 완료하고 나서 load가 완료될 때까지 해당 thread를 sema\_down을 통해 기다리게 한다. Semathread->exit\_status가 -1이고, semathread->fd가 -5이면, process\_wait(tid)를 return한다. 만약, semathread->fd 가 -5가 아니라면 tid를 return한다.**

* **bool load(const char \*file\_name, void (\*\*eip) (void), void \*\*esp)**

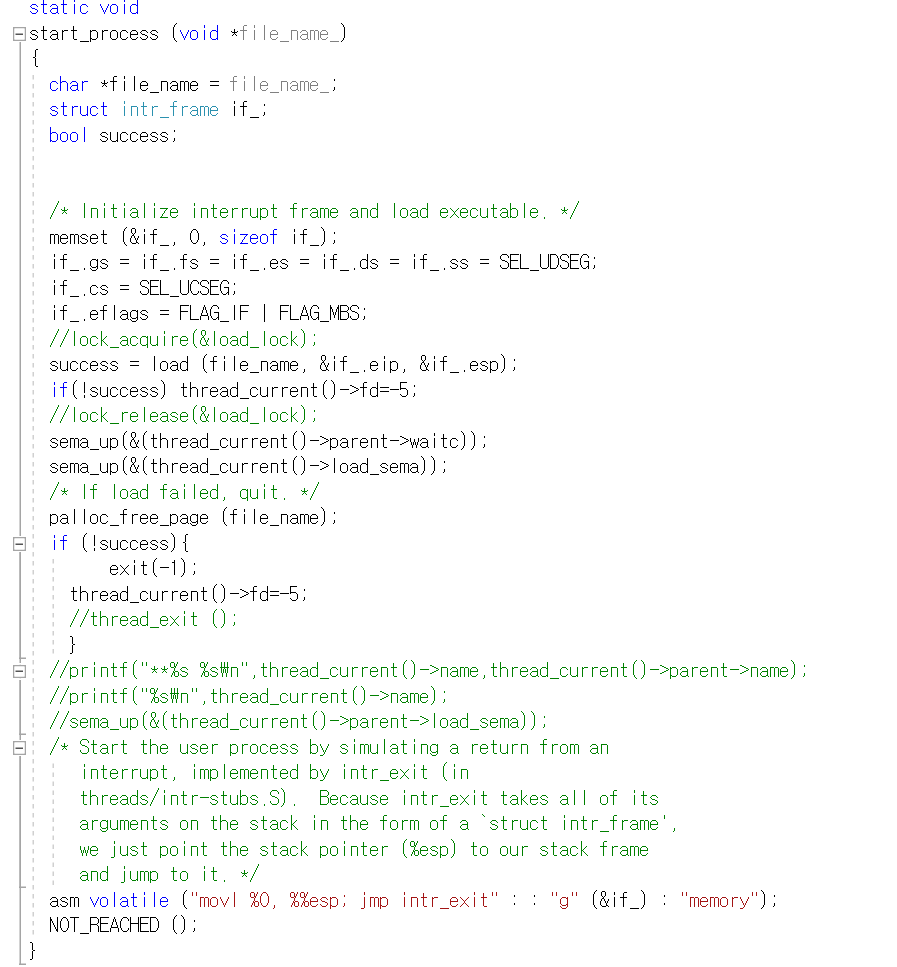
****

****

**\* 함수 설명 : file\_open을 통해, 파일을 열고, file에 저장한다. File이 NULL이라면, 락을 해제해주고, thread 구조체의 run\_file을 현재 실행할 파일로 초기화한다.   
file\_deny\_write를 통해 파일에 대한 write()를 거부한다. 그 후에 done으로 가서 , file을 닫아준다.**

**\* 사용변수 :  
\* 내 용 : 성공적으로 load가 완료되면 file을 닫고, success를 return한다.**

**- Static void start\_process(void \*file\_name\_)**

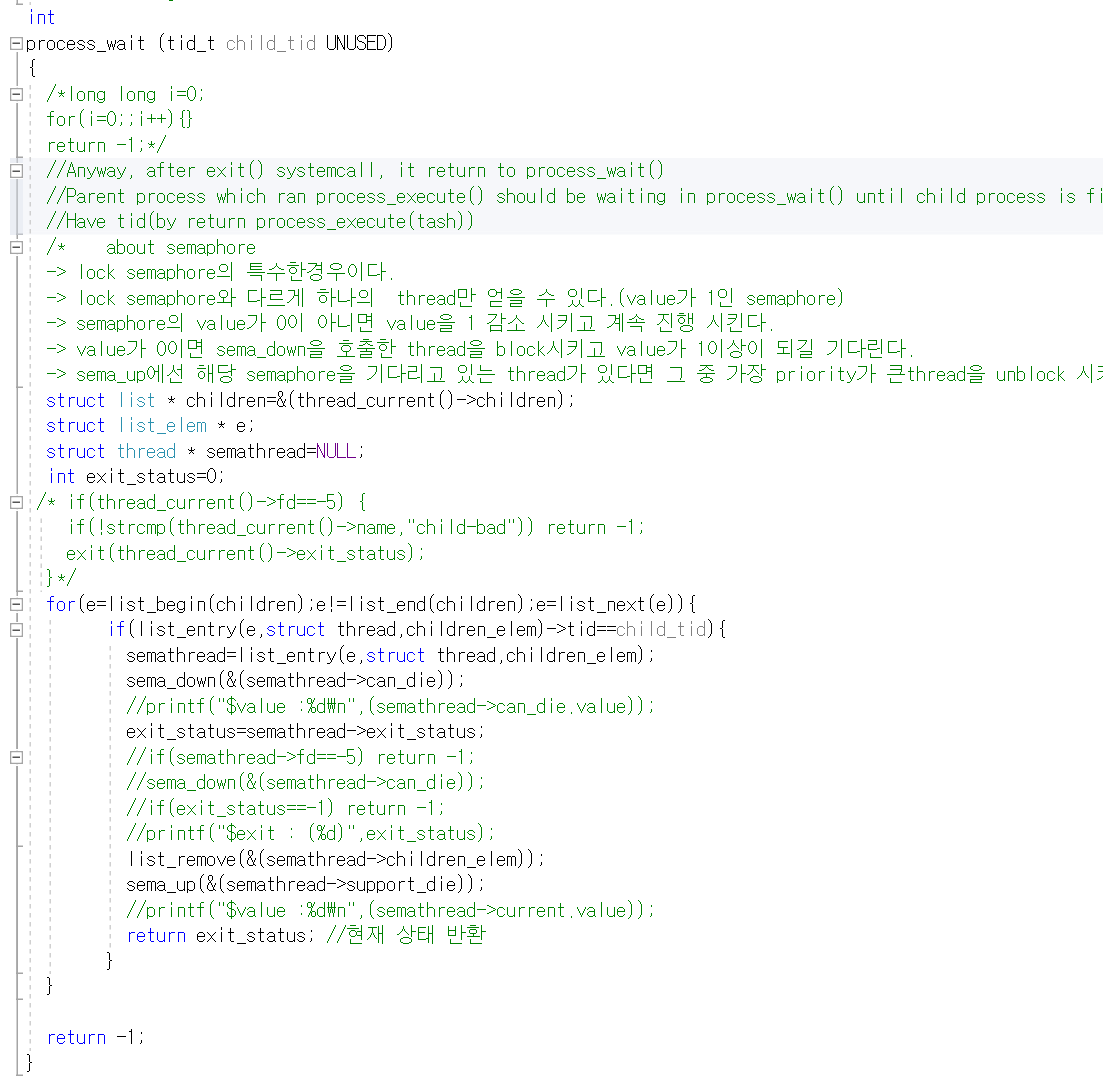
****

**\* 함수 설명 : child thread에 대한 load가 완료되면 sema\_up(load\_sema)을 통해 exec함수에서 기다리고 있는 thread에 알려준다.**

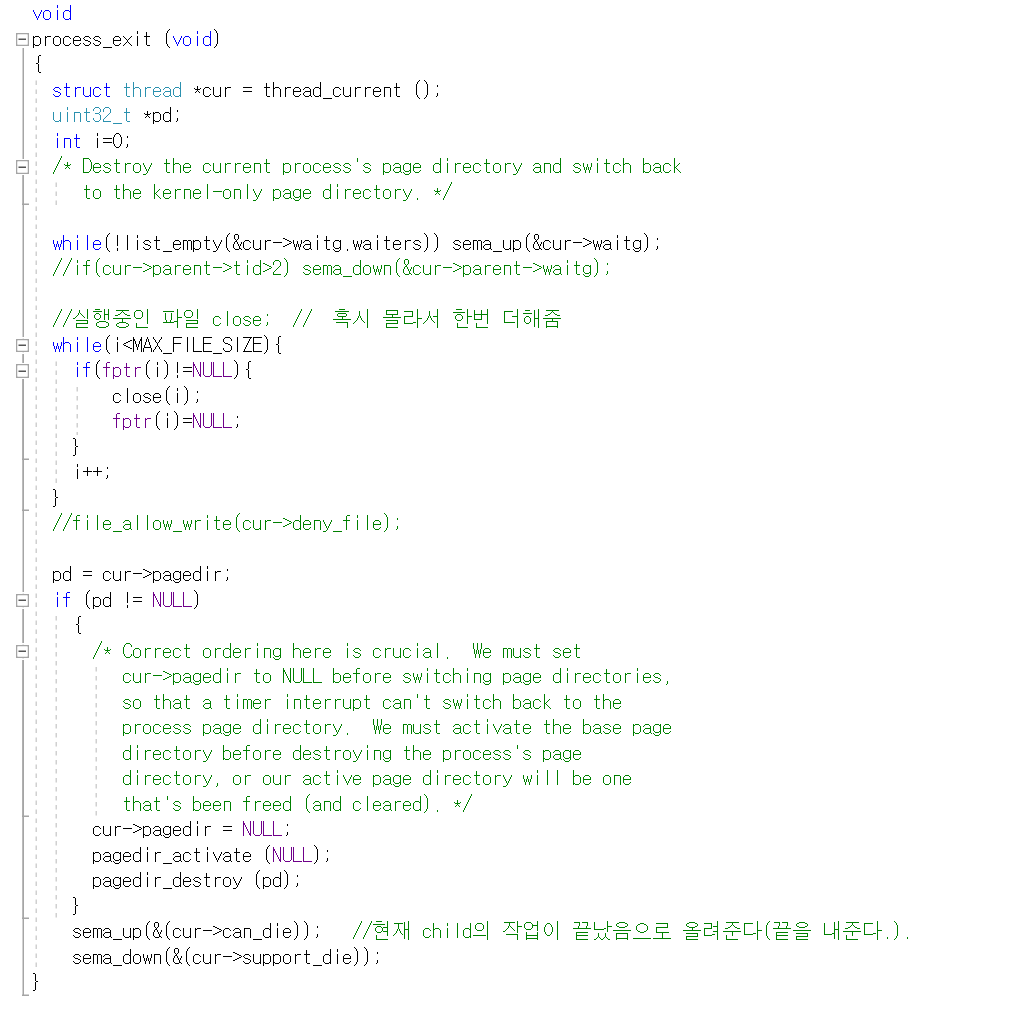
**\* 사용변수 :  
bool success : file\_name 함수를 통해 수행 된 값을 return 받는다.**

**\* 내용: 성공적으로 load가 완료되면 sema\_up을 통해 exec에서 기다리고 있는 thread에 알려줘서 block을 풀어준다.**

**- int process\_wait(tid\_t child\_tid)**

**  
\* 함수 설명 : 실질적으로 system call이 wait일 때 실행되는 함수이다. thread wait가 일어나기 위해서는 반드시 필요한 함수이고, thread 상호간에 실질적인 동기화가 일어나는 함수이다.   
\* 사용변수 :  
int exit\_status : exit되는 child thread의 status를 저장하는 변수  
\* 내 용 : parent thread가 wait에 들어오면 child thread가 exit될떄까지 sema\_down을 통해 기다리고 child thread가 exit되면 status에 child thread의 status를 저장한 후 sema up을 통해 thread를 완전히 exit한다. 그 후에 exit\_status를 return한다. 또한, 이 함수는 lock semaphore의 특수한 경우이다. Lock semaphore와 다르게 하나의 thread만 얻을 수 있다.(value가 1인 semaphore). Semaphore의 value가 0이 아니면, value를 1을 감소시키고, 계속 진행 시킨다. Value가 0이면 sema\_down을 호출한 thread를 block시키고 vaule가 1이상이 되길 기다린다. Sema\_up에선 해당 semaphore을 기다리고 있는 thread가 있다면 그 중 가장 priority가 큰 thread를 unblock 시키고, value를 1 증가 시킨다.**

* **void process\_exit(void)**

****

**\* 함수 설명 : process를 exit하는 함수이다.**

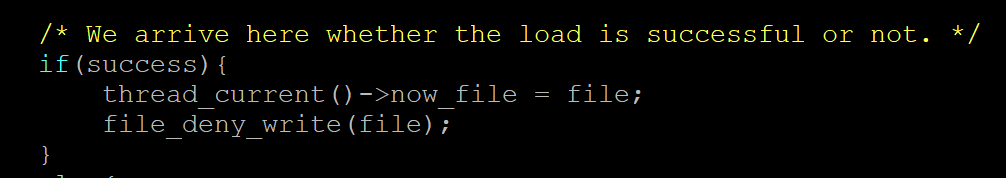
**\* 사용변수 :**

**struct thread \*cur : thread\_current()의 함수의 값을 저장한다.**

**\* 내 용 : process\_wait에서 sema\_down을 통해 parent thread가 기다리고 있고 child thread가 진행돠는데 진행이 끝나서 process\_exit에 들어오면 sema\_up을 해줘서 기다리고 있는 parent\_thread에 알려준다. 알려준 후에는 sema\_down을 통해 child\_elem을 지운후 남아있는 child thread를 말끔히 지워준다.**

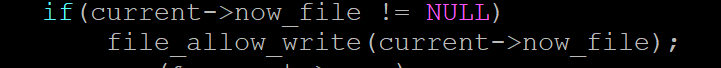
**3.Denying Writes to Executables**

* + - 1. **file\_deny\_write(file)의 활용**

****

**pintos/src/userprog/process의 load()에서 success가 true, 즉 load가 성공하였을 경우 Denying Write를 해주었다. load가 성공하면 file을 다른 thread에서 접근할 수 없도록 우선적으로 방어할 수 있도록 만든 것이다.**

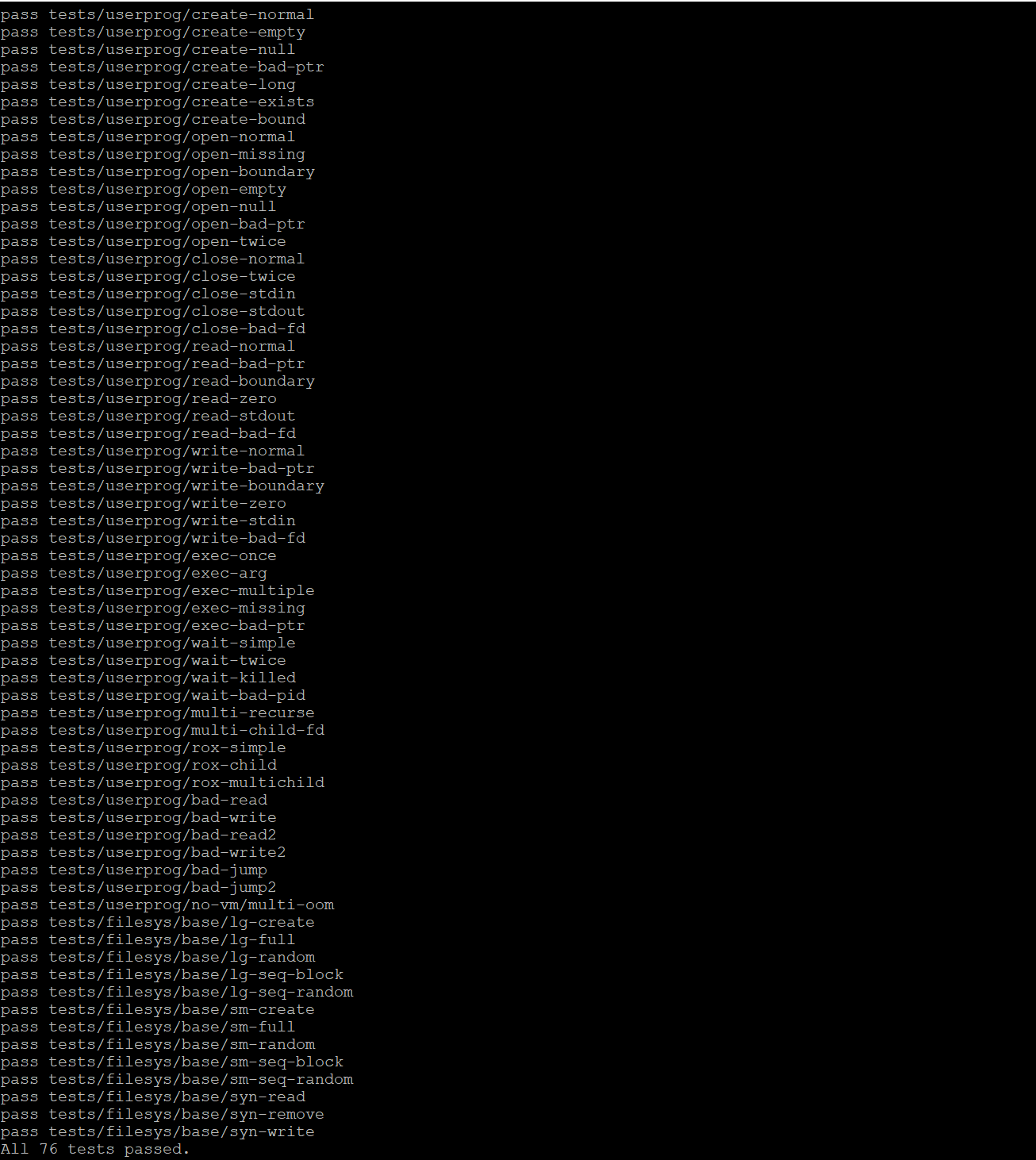
* + - 1. **file\_allow\_write(file)의 활용**

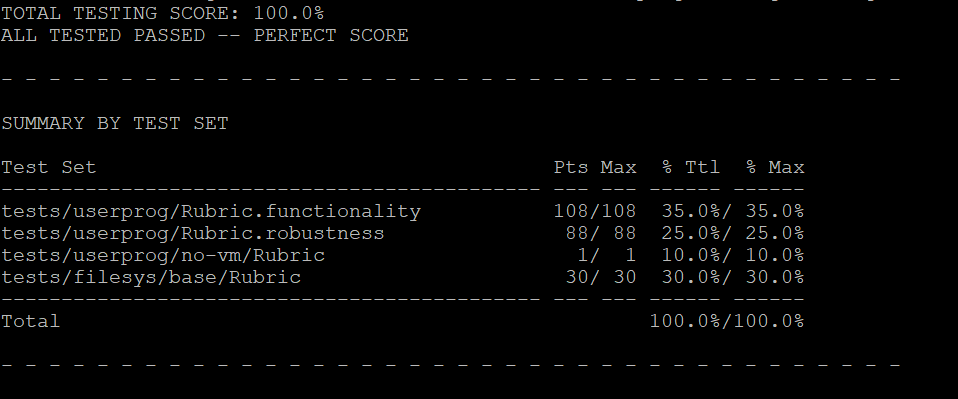
****

**pintos/src/threads/thread의 thread\_exit()에서 Denying Write를 풀어주었다. 이는 thread가 exit하는 경우에 current thread에서의 now\_file이 NULL이 아니기만 하면 file의 접근을 허락해주는 것이다.**

**C. 시험 및 평가 내용**

**- make check 결과 / make grade 결과**



****

1. **기타**
   1. **연구 조원 기여도**

* **장승민(50%), 장진영(50%)**
  1. **소감**
* **장승민 : Project 1에서는 기본적인 System call을 구현하기 때문에 크게 어려움은 느끼지 못했다. 하지만 이번 Project 2에서는 Synchronization과 multi-oom 수행을 위하여 Debugging을 시도했는데, multi-threading 환경에서 semaphore를 사용하면서 동시에 debugging 하기란 정말 어려웠다.**

**먼저 semaphore를 강의자료와 pintos manual을 정독하면서 개념 이해를 완료했다 생각했었다. 하지만 이론적인 부분과 실질적으로 적용하는 것은 명확히 달랐다. 이론적인 부분은 Critical Section, 임계지점을 확실하게 알고 있다는 전제를 두고 해야했지만, pintos에서의 임계지점을 찾는 것 부터가 쉽지 않았다. 하지만 실질적으로 file을 접근하는 부분인 write, read를 위주로 임계지점을 찾아서 semaphore를 활용해 synchronization을 완료했다. 동기화보다 multi-oom을 구현하는 것이 훨신 더 힘들었다.**

**multi-oom은 메모리누수를 확인하고 계속해서 child를 생성, 제거하면서 thread memory를 바꾸어주는 test case였다. multi-oom을 구현하기 위하여 가장 큰 시간을 투자했는데, 하나의 문제가 해결되면 다른 문제가 발생했고 동기화도 상당히 어려웠다. debugging을 하는데 갑자기 exit(-1)이 발생하는 경우도 존재했고, exit status가 제대로 넘어가지 않는 경우도 많이 존재했다. 기존에 짜놓은 코드가 모두 정확한 것이 아니였다는 것을 크게 느끼면서 semaphore를 활용하면서 충돌하는 경우도 존재하였기 때문에 이를 모두 확인하기가 너무 어려웠고 시간이 오래 걸렸다. 하지만, 결국 많은 시행 착오 속에, multi-oom을 구현했고, 그 결과 프로젝트에서 요구하는 모든 사항을 구현할 수 있어 매우 뜻 깊은 경험이었다.**

* **장진영: 이 프로젝트를 진행하면서 file을 어떻게 여는지, 실행시키는지, 닫는지 등에 대한 전반적인 지식을 실제로 경험해볼 수 있었다. 이를 통해 파일 시스템에 대해 좀 더 깊숙한 이해를 할 수 있었고, 다양한 문제 상황에서의 해결 능력을 기를 수 있었다.**

**특히 동기화 관련하여 syn-write와 syn-read 를 진행하면서, 프로세스들이 어떤 식으로 생성되고 어떤 부분에서 동기화를 시켜줘야 할 지 찾지를 못해서 많이 애를 먹었다. 며칠동안 코드를 분석하고 프로세스의 이름과 tid를 출력해가면서, 확인을 했고 세가지의 semaphore 를 적절하게 이용함으로써 완성을 하였다.**

**팀원과 의견을 나누면서도 같은 코드에 대해 의견이 갈린 적도 있지만, 어떻게 진행되는지 명확히 분석하고 문제점을 찾아가면서 배우는 것도 많았다. 내가 부족한 부분에 대해, 그리고 팀원이 부족한 부분에 대해 서로 맞춰 가면서 서로가 알아낸 것을 공유하고, 수정하며 정보를 쌓아 나갔기에 여기까지 왔다고 생각을 한다. 너무나 뜻 깊은 프로젝트였던 것 같다.**