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

담당 교수 / 분반 : 박성용 교수 / CSE4070-01

이름 / 학번 : 강민석 / 20181589

개발 기간 : 2022.10.04 ~ 2022.10.11

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

이번 프로젝트의 목표는 file system과 관련된 system call을 구현하는 것이다. 여기서 file system이란 file과 directory를 관리하는 커널의 모듈이다. 이 system call에는 다음이 해당된다. create, remove, open, close, filesize, read, write, seek, tell. 이를 위해서는 먼저 base file system을 이해해야 한다.

**개발 범위 및 내용**

* 1. **개발 범위**
* 아래 각 항목을 구현해야 하는 이유, 혹은 구현 시 기대되는 결과를 간략히 서술

1. File Descriptor

File Descriptor란 프로세스가 file을 다룰 때 사용하는 개념으로, 프로세스에서 특정 파일에 접근할 때 사용하는 정수 값이다. Pintos에서도 모든 객체를 파일로 관리한다. 즉, pintos에서 제공하는 file system 관련 함수를 이용하기 위해서는 file descriptor가 필요하다. 우리는 2 이상의 양의 정수에 해당하는 fd 값을 이용할 수 있으며, 0은 STDIN, 1은 STDOUT으로 기본적으로 할당되어 있다. (원래 리눅스에는 2가 STDERR로 할당되어 있지만, pintos manual을 보면 STDERR는 고려하지 않는다고 되어 있다.)

2. (이번 프로젝트에서 구현해야 하는) System Calls

user program에서 file system을 사용할 수 있게 system call에 file system function들을 syscall.c에 추가한다. create(파일 생성), remove(파일 제거), open(파일 열기), close(파일 닫기), filesize(파일 크기 반환), read(파일 읽기), write(파일 쓰기), seek(파일 탐색), tell(파일에서 현재 offset을 반환)

3. Synchronization in Filesystem

여러 child 간의 synchronization을 위해 semaphore를 사용하고, file 접근 synchronization을 위해 file lock을 사용한다. 어떠한 shared data를 이용하기 위해 여러 thread가 경쟁(race)를 하는데, 어떠한 프로세스가 read/write하는 data에 대하여 또다른 프로세스가 read/write하게 되면 큰 문제가 발생할 수 있다. 이를 위해서 critical section을 필히 보호해 줘야 한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

struct file 구조체의 file \*를 이용해서 struct thread에 struct file \* fd[128];을 추가했다. 원래는 struct file 구조체가 filesys 폴더 안에 file.c에 선언되어 있었는데, 여러 파일에서도 import해서 사용하기 위해 file.h로 이동했다. thread가 생성이 되면 fd[128] 배열은 모두 NULL로 초기화가 된다. 배열, linked list 등 여러 옵션이 있었지만 fd table을 배열로 만든 이유는 linked list로 만들면 메모리 관리(free 등)가 힘들 것이라고 생각해서이다.

2. System Calls: 구현할 각 system call에 대해 간략히 서술 (하나의 system call 당 최대 3문장으로 간략히 설명; 3문장을 넘길 정도로 길게 작성하지 말 것)

- SYS\_CREATE: file을 생성하는 함수이다. filesys\_create를 사용해 구현한다.

- SYS\_REMOVE: file을 삭제하는 함수이다. filesys\_remove를 사용해 구현한다.

- SYS\_OPEN: filesys\_open을 사용해 특정 file을 연다. 성공하면 fd값을 반환한다.

- SYS\_FILESIZE: file\_length를 사용해 file의 크기를 반환한다.

- SYS\_WRITE: fd값이 1인 경우 STDOUT이므로 putbuf() 함수를 사용하여 출력한다. fd가 2 이상일 경우에는 file\_write를 이용해 구현하고, lock을 이용해서 synchronization한다.

- SYS\_READ: fd값이 0인 경우 STDIN이므로 input\_getc() 함수를 사용하여 입력을 받는다. fd가 2 이상일 경우에는 file\_read를 이용해 구현하고, lock을 이용해 synchronization한다.

- SYS\_SEEK: file\_seek를 사용해 구현한다. fd에 해당하는 특정 file의 위치를 position으로 바꿔준다.

- SYS\_TELL: file\_tell을 사용해 구현한다. fd에 해당하는 특정 file의 현재 offset 위치를 반환한다.

- SYS\_CLOSE: file\_close를 사용해 구현하며, fd에 해당하는 특정 file을 닫는다.

3. Synchronization in Filesystem: Lock, Semaphore를 어떻게 이용할 수 있는지 각각에 대해 설명 (다른 방법을 서술해도 되지만 lock과 semaphore는 반드시 포함해야 함)

총 세 가지의 lock, semaphore를 사용하였다.

첫번째는, running program에 대해서 executable file이 수정되지 않도록 해야 하므로 open과 write에서 file\_deny\_write을 이용해 제한했다.

두번째로, 각각의 child에게 wait 명령을 하기 전까지 parent가 죽으면 안되는데, child thread가 load되기 전에 parent thread가 죽어버리면 애초에 실행이 안되는 문제가 생기기 때문이다. 그러므로 sema\_up과 sema\_down을 사용하여 child의 load가 끝난 후에 parent thread가 실행을 하도록 해주었다.

세번째로, file을 처리할 때 여러 프로세스가 동시에 처리하는 것을 막기 위해 lock\_acquire과 lock\_release를 사용해 구현했다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

10.04 ~ 10.05 : 프로젝트 요구사항 확인 및 개념 공부

10.06 ~ 10.10 : system call 구현 (synch 포함)

10.11 : 보고서 작성

* 1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드
  + 수정하거나 추가해야 하는 자료구조
  + 수정하거나 추가해야 하는 함수

1. File Descriptor

threads/thread.h에서 struct thread 구조체에 struct file \* fd[128]; 로 fd table을 추가한다. 이후 threads/thread.c에서 init\_thread에서 thread가 생성되면 fd table을 NULL로 초기화한다.

1. System calls

userprog/syscall.c에서 syscall\_handler()의 switch문 안에 새로운 file system을 적절히 추가한다. (project 1에 이어서) create(), remove(), open(), close(), filesize(), read(), write(), seek(), tell()을 구현하고, 이에 맞게 userprog/syscall.h를 수정한다. 또한, userprog/exception.c에서 page\_fault()의 일부분을 오류에 맞게 조금 더 강화해 주었다.

1. Synchronization in Filesystem

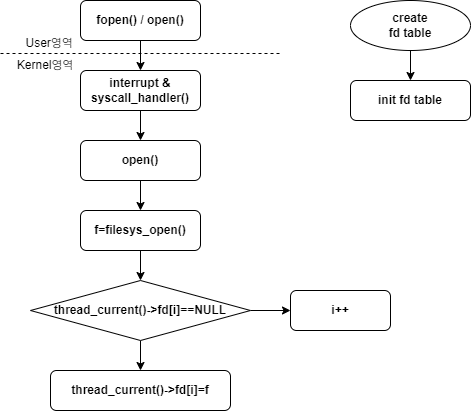
첫번째는, running program에 대해서 executable file이 수정되지 않도록 해야 하므로 open과 write에서 file\_deny\_write을 이용해 제한했다. open()에서 filesys\_open을 이용해 file pointer를 return 받고, 가장 작은 빈(NULL인) fd값을 open하려는 file에 할당해 준다. 할당 직전에 file\_deny\_write를 이용하여 현재 open되어 있는 file을 수정하지 못하도록 하였다. write()에서도 write하려는 해당 fd값의 file이 deny\_write가 되어있지 않다면 file\_deny\_write를 해준다. file\_deny\_write는 file이 close될 때, filesys/file.c의 file\_close 함수에서 file\_allow\_write로 해제가 되므로 따로 적어줄 필요가 없었다.

두번째는, child의 load가 끝난 후에 parent thread가 실행을 하도록 해주기 위해 semaphore를 사용했다. 먼저 threads/thread.h의 struct thread에 struct semaphore lock\_load;를 해주었고, threads/thread.c의 init\_thread()에서도 lock\_load를 sema\_init해 주었다. 이후 userprog/process.c의 process\_execute (parent에 해당)에서 thread\_create를 통해 child를 만들고, 그 child의 lock\_load를 통해 sema\_down을 해준다. child는 start\_process()에서 load를 마치고 나서 본인의 lock\_load를 통해 sema\_up을 해주고, 그러면 parent는 다시 코드를 수행한다. 이렇게 하면 child가 load되기 전에 parent가 종료되는 것을 방지할 수 있다.

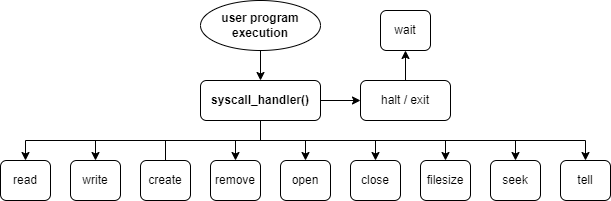
마지막으로, file을 처리할 때 여러 프로세스가 동시에 처리하는 것을 막기 위해 lock\_acquire과 lock\_release를 사용해 구현했다. 이 함수들은 open(), read(), write() 함수가 시작할 때와 끝날 때에 각각 lock\_acquire, lock\_release를 사용했다.

1. **연구 결과**
   1. **Flow Chart**

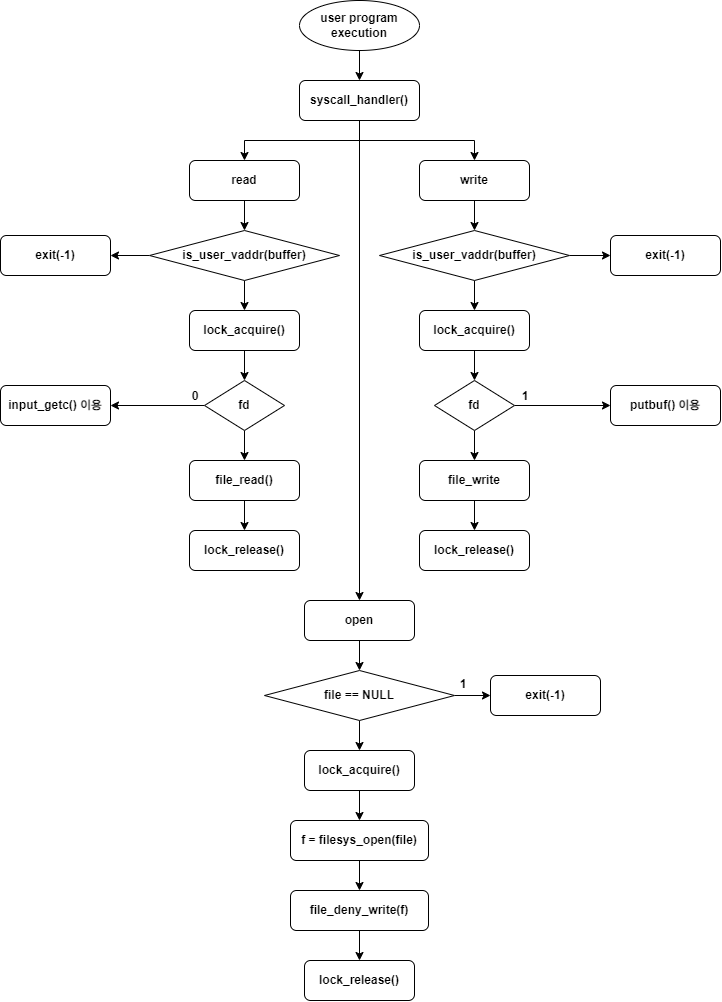
* II. B. 개발 내용의 각 3가지 항목에 대하여 Flow Chart 작성
  + 1. File Descriptor



* + 1. System calls



* + 1. Synchronization in Filesystem



* 1. **제작 내용**
* II. B. 개발 내용의 각 3가지 항목에 대하여 실직적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
* 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명
  + 1. File Descriptor

threads/thread.h의 struct thread에 다음과 같이 file descriptor table을 추가한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

threads/thread.c의 init\_thread에서 초기화해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이후 userprog/syscall.c의 open()에서 file을 열 때마다 file descriptor를 할당해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* + 1. System calls

텍스트이(가) 표시된 사진

자동 생성된 설명userprog/syscall.h에 새로 추가할 syscall handler 함수를 prototyping해준다.

이후 system call handler에서 SYS\_CREATE, SYS\_REMOVE, SYS\_OPEN, SYS\_FILESIZE, SYS\_WRITE, SYS\_READ, SYS\_SEEK, SYS\_TELL, SYS\_CLOSE에 해당하는 함수들을 추가해 주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

- create(): file을 생성하는 함수이다. filesys\_create를 사용해 구현한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

- remove(): file을 삭제하는 함수이다. filesys\_remove를 사용해 구현한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

- open(): filesys\_open을 사용해 특정 file을 연다. 성공하면 fd값을 반환한다. 파일명을 인자로 받으며 NULL일 경우에 exit(-1)을 한다. 이후 fd table에서 빈 곳을 찾아 open한 file의 포인터를 넘겨 주고 file의 이름과 현재 실행중인 file의 이름이 같다면 file\_deny\_write()를 통해 제한한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

- filesize(): file\_length를 사용해 file의 크기를 반환한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

- seek(): file\_seek를 사용해 구현한다. fd에 해당하는 특정 file의 위치를 position으로 바꿔준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

- tell(): file\_tell을 사용해 구현한다. fd에 해당하는 특정 file의 현재 offset 위치를 반환한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

- close(): file\_close를 사용해 구현하며, fd에 해당하는 특정 file을 닫는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* exit(): 기존 exit을 조금 수정해 주었다. 프로세스를 종료할 때, NULL이 아닌 fd를 모두 close해 주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* + 1. Synchronization in Filesystem
* lock 사용
* 아래 그림과 같이 lock을 선언하고 syscall\_init에서 lock\_init을 해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* write(): fd값이 1인 경우 STDOUT이므로 putbuf() 함수를 사용하여 출력한다. fd가 2 이상일 경우에는 file\_write를 이용해 구현하고, lock을 이용해서 synchronization한다. 또한, deny\_write가 설정되어 있지 않다면, file\_deny\_write를 통해서 동시 write를 제한한다. file 접근 synchronization을 위해 맨 처음과 끝에 lock\_acquire과 lock\_release도 해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* 텍스트이(가) 표시된 사진

  자동 생성된 설명read(): fd값이 0인 경우 STDIN이므로 input\_getc() 함수를 사용하여 입력을 받는다. fd가 2 이상일 경우에는 file\_read를 이용해 구현하고, lock을 이용해 synchronization한다.
* 하지만, lock을 이용해 synchronization을 구현했음에도 불구하고 syn\_read와 syn\_write testcase가 계속해서 FAIL이 발생했다. FAIL이 발생한 부분을 확인해 보니, child의 load가 정상적으로 되지 않고 종료된 것을 볼 수 있었다. child가 load되기 전에 parent process가 종료된 것이다. 이를 해결하기 위해 lock\_load semaphore를 이용하였다. threads/thread.h의 struct thread에 struct semaphore lock\_load;를 해주었고, threads/thread.c의 init\_thread()에서도 lock\_load를 sema\_init해 주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

* 이후 userprog/process.c의 process\_execute (parent에 해당)에서 thread\_create를 통해 child를 만들고, 그 child의 lock\_load를 통해 sema\_down을 해준다. child는 start\_process()에서 load를 마치고 나서 본인의 lock\_load를 통해 sema\_up을 해주고, 그러면 parent는 다시 코드를 수행한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

* 그런데도, multi-oom을 계속해서 FAIL하였다. 메모리 누수가 발생했다는 얘기인데, 어느 부분에서 메모리 정리를 안 해줬는지 확인해 본 결과 process\_exit에서 file이 비정상 종료 등으로 인해 제대로 close되지 않은 부분이 있는 것을 알았다. 그래서 이를 수정해 주었다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

* 1. **시험 및 평가 내용**

1. make check 수행 결과를 캡처하여 첨부

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

* make grade

텍스트이(가) 표시된 사진

자동 생성된 설명