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

담당 교수 / 분반 : 김영재

이름 / 학번 : 김현규 / 20161580

개발 기간 : 20.11.9 ~ 20.11.17

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

**File을 위한 System Call들을 구현한다. 해당 System Call들은 다음과 같다.**

**(create, remove, open, filesize, read, write, seek, tell, close)**

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목을 구현해야 하는 이유, 혹은 구현 시 기대되는 결과를 간략히 서술
  + 1. File Descriptor

2. (이번 프로젝트에서 구현해야 하는) System Calls

3. Synchronization in Filesystem

1. File Descriptor

시스템으로부터 할당받은 파일을 대표하는 음이 아닌 정수값이다. process가 파일을 다룰 경우 필요하다.

* + 1. (이번 프로젝트에서 구현해야 하는)System Calls

해당 System Calls들은 File을 다루는 System Call이다. 따라서 이들을 모두 구현했을 시에, process가 원활하게 File을 다룰 수 있다.

* + 1. Synchronization in Filesystem

같은 파일을 동시에 Read, Write하는 경우가 일어나면 안된다. 따라서 Synchronization을 통해 해당 문제를 해결한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

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)

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

1. struct file의 자료 구조의 포인터 배열을 선언한다. 배열의 크기는 128이다. (pintos문서 System Calls FAQ 참고)

2.

-bool create(const char\* file, unsigned initial\_size): file의 이름을 가진 새 파일을 만든다. 파일의 크기는 initial\_size이다. 잘 만들어졌을 경우 true를, 아닌 경우 false를 return한다.

-bool remove(const char\* file): file의 이름을 가진 파일을 삭제한다. 잘 삭제했을 경우 true를, 아닌 경우 false를 return한다.

-int open(const char\* file): file의 이름을 가진 파일을 연다. 잘 열었을 경우 file descriptor를 return하고, 아닌 경우 -1을 return한다.

-int filesize(int fd): file descriptor가 fd인 파일의 크기를 return한다.

-void seek(int fd, unsigned position): file descriptor가 fd인 파일에서 position에 위치한 것을 찾는다.

-unsigned tell(int fd): file descriptor가 fd인 파일의 write할, 또는 read할 위치를 return한다.

-void close(int fd): file descriptor가 fd인 파일을 닫는다.

3.

thread/synch.h에 있는 struct lock를 선언한다. void lock\_init(struct lock\*)을 통해 초기화할 수 있다, void lock\_acquire(struct lock\*)을 통해 lock을 걸 수 있다. void lock\_release(struct lock\*)을 통해 lock을 해제할 수 있다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성
  1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드

-userprog/syscall.c:

1. static void syscall\_handler(struct intr\_frame \*f): 추가해야할 system call인 create, … , close에 해당하는 system call number의 case들을 추가한다.

2. create, … , close을 실행하는 함수: system call api을 이용하여 구현한다.

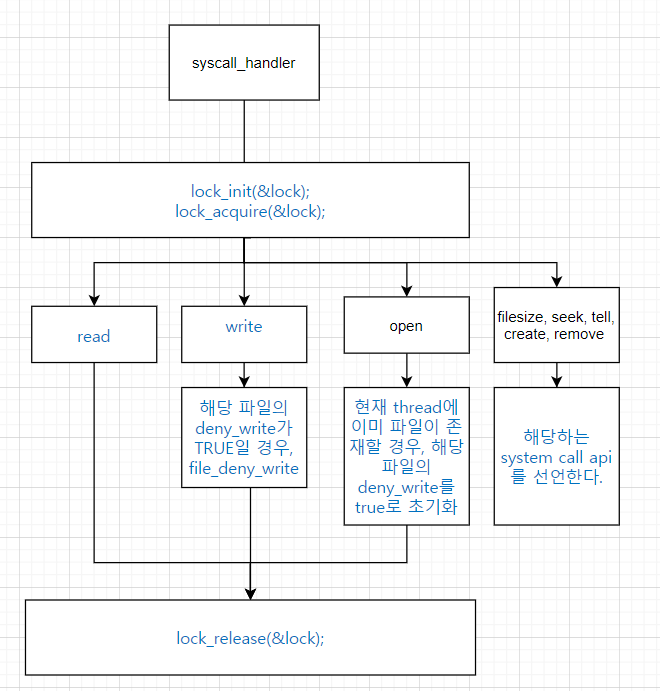
* + 수정하거나 추가해야 하는 자료구조

-thread.h: file descriptor인 struct file\* file\_des[128] 선언

-syscall.c: read, write 동기화 과정에서 필요한 lock인 struct lock fileLock 선언

1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 3가지 항목에 대하여 Flow Chart 작성



* 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 thread 선언에서 struct file\* file\_des[128] 추가. 해당 파일들은 file descriptor이다.

* 2. System Calls

다음 함수들은 모두 userprog/syscall.c에 존재하는 함수들이다.

int read(int fd, void\* buffer, unsigned size): fd가 0이 아닌 경우를 작성한다. 읽어들인 fd를 가지고 무슨 파일을 읽을지 알 수 있다. 해당 파일은 현재 돌아가는 thread(thread\_current())의 file\_des[fd]를 통해 접근 가능하다. 해당 파일이 NULL, 즉 file descriptor가 fd인 파일이 존재하지 않을 경우 exit(-1)을 호출한다.

int write(int fd, const void\* buffer, unsigned size): fd가 1이 아닌 경우를 작성한다. 읽어들인 fd를 가지고 무슨 파일에 쓸지 알 수 있다. 해당 파일은 현재 돌아가는 thread(thread\_current())의 file\_des[fd]를 통해 접근 가능하다. 해당 파일이 NULL, 즉 file descriptor가 fd인 파일이 존재하지 않을 경우 exit(-1)을 호출한다. Executable files에 write하는 것을 막기 위해 filesys/file.c의 file\_deny\_write 함수를 사용한다.

bool create(const char \*file, unsigned initial\_size): system call api인 filesys\_create(file, initial\_size)를 통해 크기가 initial\_size인 파일을 만든다.

bool remove(const char \*file): system cal api인 filesys\_remove(file)를 통해 해당 이름의 file을 삭제한다.

int open(const char \*file): system call api인 filesys\_open(file)을 통해 해당 이름의 file을 연다. 해당 파일은 현재 돌아가는 thread의 file descriptor table에 할당해줘야 한다.(thread\_current()->file\_des[i] = f, f는 파일을 가리키는 변수이다.)

int filesize(int fd): 입력받은 fd를 바탕으로 file descriptor table에서 해당 파일을 가져올 수 있다. 이후 system call api인 file\_length를 이용함으로써 파일의 크기를 알 수 있다.

void seek(int fd, unsigned position), unsigned tell(int fd): 입력받은 fd를 바탕으로 file descriptor table에서 해당 파일을 가져올 수 있다.(thread\_current()->file\_des[fd]) system call api인 file\_seek, file\_tell을 이용하여 원하고자 하는 바를 실행할 수 있다.

void close(inr fd): 입력받은 fd를 바탕으로 file descriptor table에서 해당 파일을 가져올 수 있다.(thread\_current()->file\_des[fd]) system call api인 file\_closel을 이용하여 원하고자 하는 바를 실행할 수 있다. 그리고 해당 파일은 NULL로 초기화해준다.

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

void syscall\_init(void)에서 lock\_init(&lock)으로 lock을 초기화한다. 그리고 control section의 앞 뒤에(read함수의 시작-끝, write함수의 시작 끝) lock\_acquire(&lock), lock\_release(&lock)을 통해 동기화한다.

* 1. **시험 및 평가 내용**
* make check 수행 결과를 캡처하여 첨부 