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

담당 교수 / 분반 : 김영재 / 2

이름 / 학번 : 조린 / 20171692

개발 기간 : 2주

1. **개발 목표**

File system에 관해 pintos에서 구현되어있는 기능들을 user program에서 system call을 통해 사용할 수 있도록 하고, synchronization을 통해 shared data에 생길 수 있는 에러를 방지한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

1. File Descriptor

process가 개별적인 file descriptor table을 가져 현재 사용하고 있는 파일들을 저장할 수 있도록 한다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

Create - user program으로 file을 생성할 수 있다.

Remove - user program으로 file을 제거할 수 있다.

Open - user program에서 file의 데이터를 사용할 수 있게 한다.

Close - user prog에서 file을 사용하는 것을 종료하고, 자원을 회수한다.

Filesize - file의 크기를 반환한다.

normal read - standard input만이 아니라 file에서도 데이터를 읽어와 buffer에 저장한다.

normal write - standard output만이 아니라 file에도 buffer안의 데이터를 출력해 저장한다.

Seek - file을 변경 또는 읽어올 위치를 새로 정한다.

Tell - file을 변경 또는 읽어올 위치를 반환한다.

3. Synchronization in Filesystem

한 번에 한 프로세스만이 file system에 접근함으로써 critical section에 둘 이상의 process가 접근하는 것을 방지한다.

* 1. **개발 내용**

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

Simple array. file descriptor table의 크기가 정해져있고(128) fd는 int형으로 정의되는데 linked list로 구현할 경우(list in pintos) fd에 따라 파일에 접근하거나 interior element인 fd를 삭제할 때와 같은 상황에서 직관적인 구현이 아니라고 판단했고, list의 특정 원소에 접근할 때 overhead가 과하게 생기는 것이 적절하지 않다고 판단했음.

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)

Create - operand의 주소가 정확한지 확인한 후에, operand가 형식에 맞는지 확인하고, /filesys에서 구현되어있는 API를 수행한다.

Remove - user program으로 file을 제거할 수 있다.

Open - user program에서 file의 데이터를 사용할 수 있게 한다.

Close - user prog에서 file을 사용하는 것을 종료하고, 자원을 회수한다.

Filesize - file의 크기를 반환한다.

normal read - standard input만이 아니라 file에서도 데이터를 읽어와 buffer에 저장한다.

normal write - standard output만이 아니라 file에도 buffer안의 데이터를 출력해 저장한다.

Seek - file을 변경 또는 읽어올 위치를 새로 정한다.

Tell - file을 변경 또는 읽어올 위치를 반환한다.

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

filesystem에 접근할 때 lock을 통해 다른 프로세스들이 접근하지 못하도록 한 뒤 원하는 데이터를 처리하고 lock을 풀어서 동시에 여러 개의 프로세스가 데이터를 처리하지 못하도록 한다.

file을 읽는 것은 데이터의 변경이 일어나지 않기 때문에 여러 개의 프로세스가 동시에 접근을 해도 괜찮은데, 이를 이용해서 write만 semaphore에 의해 데이터에 접근하지 못하도록 하는 방법이 있다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

10/3 - syscall.c에서 새로 구현하는 system call 함수들의 기본적인 틀 작성

10/4 ~ 6 - 추가한 system call들의 예외처리

10/7 - struct thread에서 file descriptor table 추가 및 fd\_table을 사용하는 함수들을 완성

10/10 - lock을 통해 filesystem synchronization 구현

10/11 ~ 12 - 실패하는 test case를 검사하고 코드 수정

* 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

File descriptor - threads에서 thread 구조체에 fd\_table array추가 및 NULL로 초기화하는 코드 추가. fd를 변수로 받는 system call 함수(filesize, seek, tell, close, read, write)에서 fd\_table을 통해 file에 접근하도록 하는 코드 추가. process\_exit에서 프로세스 종료 시 열려있는 file을 반복문을 통해 닫아주는 코드 추가.

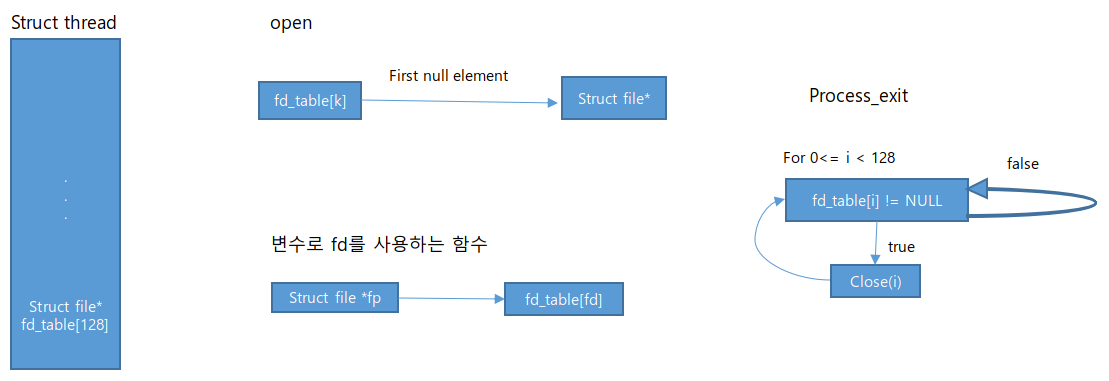
System call - syscall.c에서 system call과 kernel API를 연결해주는 함수들 추가(create, open, close, …), syscall\_handler에 case 추가. esp의 주소 및 내용물이 형식에 맞는지 확인하는 코드 추가.

Synchronization in filesystem - system call을 처리하는 과정에서 filesystem에 한 프로세스만 접근할 수 있도록 하기위한 lock 추가. syscall\_init 에서 lock도 같이 초기화하도록 코드 추가. read, write함수를 수행하기 전에 lock을 얻고 수행이 끝난 뒤 lock을 풀어주는 코드 추가.

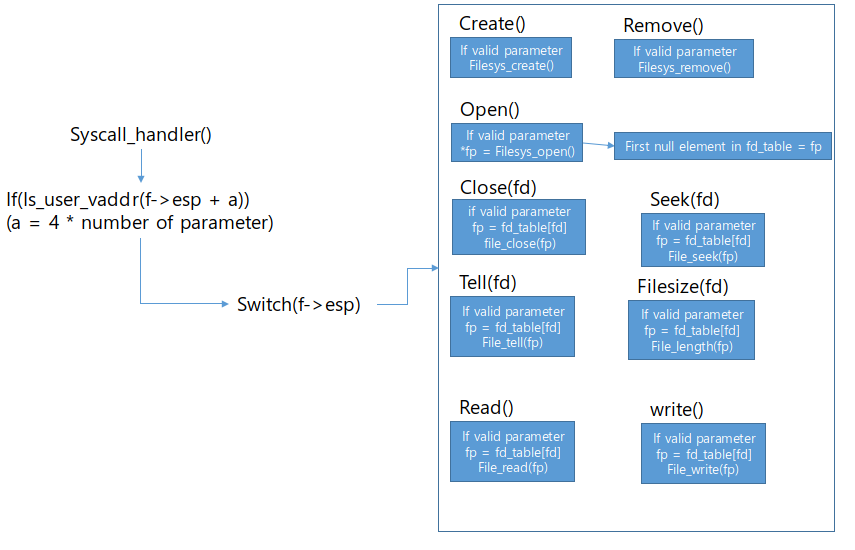
1. **연구 결과**
   1. **Flow Chart**

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

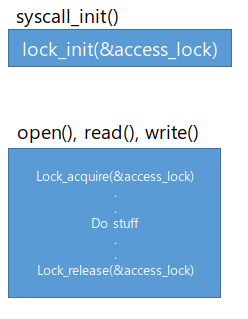
1. file descriptor



2. system call



3. 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.h에 fixed array fd\_table[128]을 추가함.

Thread.c의 Init\_thread()에서 fd\_table의 원소를 반복문을 통해 접근해서 NULL로 초기화함.

Userprog/syscall.c의 filesize(), seek(), tell(), close()에서 현재 프로세스의 fd\_table의 fd번째 원소에 저장되어있는 file pointer를 가져와 필요한 함수를 수행함.

Process.c의 process\_exit()에서 fd\_table의 원소를 탐색하는 반복문을 통해 NULL이 아닌 원소가 있다면 close()를 호출해 정상적으로 닫아줌으로써 자원을 회수함.

2. system call

Userprog/syscall.c의 syscall\_handler에서 이번에 구현하는 system call인 create, remove, open, close, seek, tell, filesize에 대한 case를 추가함. 각 case에 대해 변수가 저장되어있는 주소가 user stack인지 확인하고, 아니라면 종료한다.

create() - filesys/filesys.h에 정의되어있는 kernel API인 filesys\_create()를 통해 변수로 넘어온 name을 가지고 파일을 생성함.

remove() - filesys/filesys.h에 정의되어있는 kernel API인 filesys\_remove()를 통해 변수로 넘어온 name을 가지고 파일을 제거함.

open() - filesys/filesys.h에 정의되어있는 kernel API인 filesys\_open()를 통해 변수로 넘어온 name을 가지고 파일을 불러와 struct file \*fp에 저장한다. 그 뒤 현재 프로세스의 fd\_table을 반복문을 통해서 참조해 파일이 저장되어있지 않은 가장 index가 낮은 원소에 불러온 파일을 저장한다. 만약 parent process를 불러오려고 하면 file\_deny\_write()를 통해 읽어오는 것만 가능하게 한다.

filesize() - 변수로 넘어온 fd를 통해서 얻은 파일에 대해 filesys/file.h에 정의되어있는 kernel API인 file\_length()를 통해 크기를 byte단위로 반환함.

seek() - 변수로 넘어온 fd를 통해서 얻은 파일에 대해 filesys/file.h에 정의되어있는 kernel API인 file\_seek()를 통해 파일의 offset을 변경함.

tell() - 변수로 넘어온 fd를 통해서 얻은 파일에 대해 filesys/file.h에 정의되어있는 kernel API인 file\_tell()를 통해 파일의 offset을 반환함.

close() - 변수로 넘어온 fd를 통해서 얻은 파일에 대해 filesys/file.h에 정의되어있는 kernel API인 file\_close()를 통해 프로세스와 파일의 연결을 끊고 fd\_table[fd]도 NULL로 변경해준다.

3. synchronization in filesystem

Threads/synch.h에 선언되어있는 struct lock을 통해 filesystem에 접근중인 프로세스가 있는지를 알 수 있게 한다.

Userprog/syscall.c에 filesystem으로의 접근을 동시에 하나의 프로세서만 가능하도록 조절하기 위해 struct lock access\_lock을 전역변수로 선언함.

Syscall\_init()에서 access\_lock을 lock\_init()을 통해 초기화해준다.

Open(), read(), write()에서 함수가 시작할 때 lock\_acquire()를 통해 현재 프로세스가 lock을 획득해 filesystem에 접근중임을 알리고 함수가 종료되기 전에 lock\_release()를 통해 현재 프로세스가 filesystem과의 연결을 종료했음을 알린다.

* 1. **시험 및 평가 내용**
* 