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

담당 교수 / 분반 : 김영재 / 1반

이름 / 학번 : 허재성 20161663

개발 기간 : 2021.10.01 ~ 2021.10.16

1. **개발 목표**

Project 1에서 기본적인 Argument Passing, User stack, System call, System call handler등을 구현하였다. 이에 추가적으로 기본적인 file system을 구현하여 Pintos가 파일 생성, 제거, 파일 열기, 닫기 등의 기능을 수행할 수 있도록 한다. 또한 Proj1에서 구현한 STDIN read와 STDOUT write 외에 file 입출력이 가능하도록 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

1. File Descriptor

File Descriptor는 특정 파일에 접근하기 위한 정수값으로 Thread마다 File descriptor를 가지도록 구현하면 해당 thread가 여러 파일에 접근 가능하며 서로 다른 파일에 구분하여 접근 가능하다. 기본적으로 0은 콘솔 입력인 STDIN, 1은 콘솔 출력인 STDOUT, 2는 오류 메시지 출력을 위한 STDERR로, 3 이상의 정수부터 특정 파일에 접근하는데 사용된다. 기본적인 파일 시스템(파일 열기, 닫기, 읽기, 쓰기 등)을 구현하기 위해 반드시 필요하다.

1. System call

Proj2에서 추가적으로 구현한 system call은 create, remove, open, filesize, seek, tell, close이다. 이를 구현해 pintos에서 기본적인 filesystem 기능을 수행할 수 있다. Proj1에서 구현한 read, write에 추가적으로 FILE read, write를 구현하였다.

1. Synchronization in Filesystem

파일을 여러 Process에서 동시에 읽거나 동시에 쓰는 것을 방지하기 위해 lock을 이용해 Synchronization을 구현한다. Read, write 부분이 critical section으로 한 Process가 이를 수행할 동안 다른 Process가 이를 수행해서는 안된다. 따라서 이를 보호하기 위해 critical section을 lock으로 protect하는 것을 구현한다.

* 1. **개발 내용**

1. File Descriptor 구현

Pintos manual에 따라 threads/thread.h의 thread 구조체에 file descriptor를 위한 file 포인터 배열을 fd를 선언하였다. 배열의 크기는 manual에 따라 128로 하였으며 thread에서 파일을 열 때 3번부터 127번까지(0, 1, 2는 각각 STDIN, STDOUT, STDERR 로 예약됨) 중에 아직 할당되지 않은 번호를 할당하도록 했다.

2. System call

create : 인자로 주어지는 name을 파일 이름으로, initial\_size만큼의 크기를 가지는 파일을 생성한다. 이를 위해 filesys/filesys.c의 filesys\_create 함수를 호출하여 구현한다. 생성 성공 시, true, 실패 시 false를 반환한다.

remove : 인자로 주어지는 name이란 이름의 파일을 삭제한다. 이를 위해 filesys/filesys.c의 filesys\_remove 함수를 호출해 구현한다. 삭제 성공 시, true, 실패 시 false를 반환한다.

open : 인자로 주어지는 file을 이름으로 가지는 파일을 연다. 이를 위해 filesys/filesys.c의 filesys\_open 함수를 호출해 해당 file의 포인터를 반환 받고 file descriptor를 반환한다.

file\_size : 인자로 주어진 file descriptor fd에 해당하는 파일의 바이트 단위 크기를 반환한다. filesys/file.c의 file\_length 함수를 호출하여 크기를 반환받고 반환한다.

seek : 인자로 주어진 file descriptor fd에 해당하는 파일의 현재 위치(file 구조체의 pos)를 인자로 주어진 position으로 수정한다. filesys/file.c의 file\_seek 함수를 호출하여 수행한다.

tell : 인자로 주어진 file descriptor fd에 해당하는 파일의 현재 위치(바이트)를 반환한다. filesys/file.c의 file\_tell 함수를 호출하여 해당 값을 반환 받는다.

close : 인자로 주어진 file descriptor fd에 해당하는 file을 닫는다. filesys/file.c의 file\_clsoe 함수를 호출하여 수행한다. 이를 위해 현재 process(thread)에서 fd에 해당하는 파일을 찾아 포인터를 file\_read에 넘겨줘야 한다.

Proj1에 추가 구현

read : 인자로 주어지는 fd가 3 이상일 경우, 인자로 주어지는 fd에 해당하는 file로부터 size 바이트만큼 읽어 들여 buffer에 저장한다. filesys/file.c의 file\_read 함수를 호출하여 실제로 읽어들인 바이트 수를 반환 받아 이를 반환한다. 현재 process(thread)에서 fd에 해당하는 파일을 찾아 포인터를 file\_read에 넘겨줘야 한다.

write : 인자로 주어지는 fd가 3 이상일 경우, 인자로 주어지는 buffer에 저장된 바이트 수만큼 fd에 해당하는 file에 쓴다. filesys/file.c의 file\_write 함수를 호출하여 실제로 쓴 바이트 수를 반환 받아 이를 반환한다. 현재 process(thread)에서 fd에 해당하는 파일을 찾아 포인터를 file\_write에 넘겨줘야 한다.

3. Synchronization in Filesystem

Critical section인 system call read, write, 그리고 open을 보호하기 위해 각 system call을 구현한 함수 my\_read, my\_write, my\_open 함수의 critical section읠 lock으로 감쌌다. Critical section 코드 직전에 lock\_acquire로 process가 lock을 얻어 critical section에 진입하고 lock을 얻지 못한 다른 process는 해당 영역에 진입할 수 없도록 했다. Lock을 얻어 진입한 process는 critical section 수행 후 lock\_release로 lock을 반환해 다른 process가 진입 가능하도록 해준다.

**추진 일정 및 개발 방법**

* 1. **추진 일정**

2021.10.03 ~ 2021.10.09 : 명세서, manual 분석, 요구사항 분석, 코드 분석

2021.10.10 ~ 2021.10.12 : Proj2 기본 코드 작성(기본적인 system call 구현)

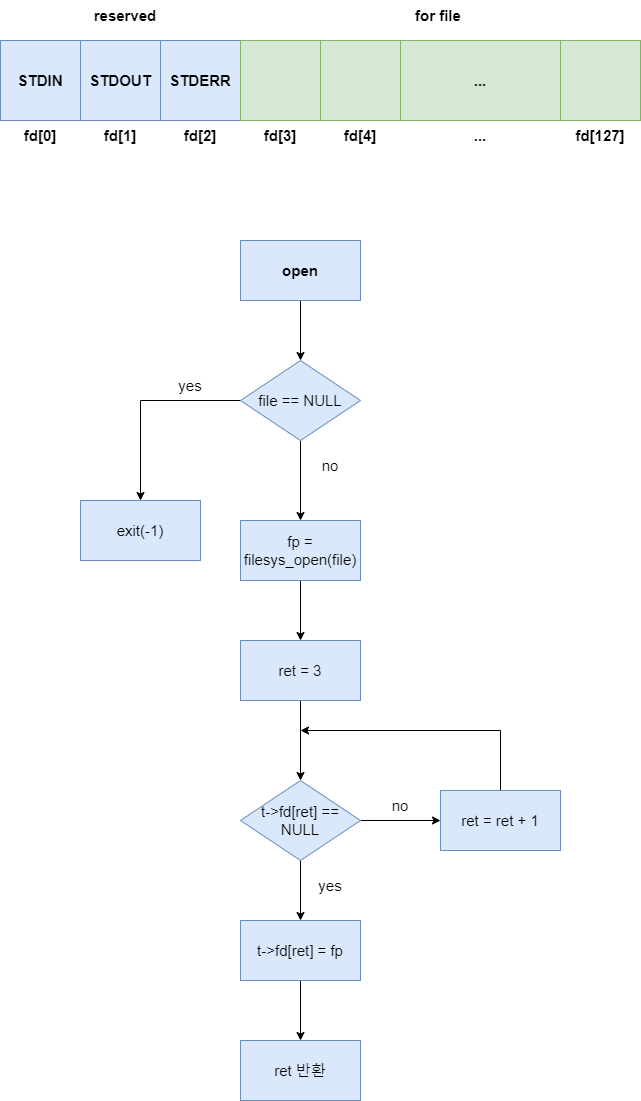
2021.10.13 ~ 2021.10.15 : Synchronization, 각종 예외처리 구현 시도

* 1. **개발 방법**

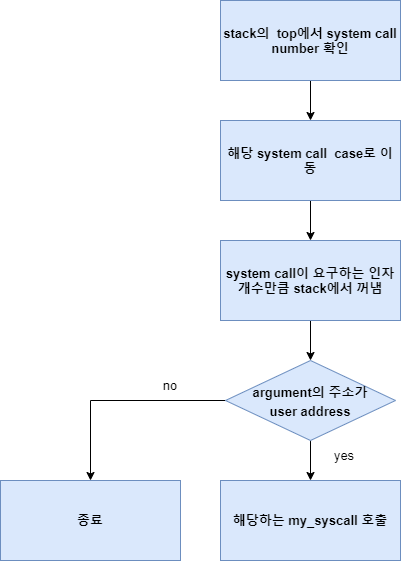
현재 프로세스가 파일 입출력을 비롯한 filesystem 기능을 수행하기 위해 file 구조체를 가져야 한다. 이를 위해 threads/thread.h의 thread 구조체에 file\* 포인터 배열을 선언했다. 크기는 manual을 참조해 128로 설정했다. Userprog/syscall.c에 Proj2에서 새로 구현하는 create, remove, open, file\_size, seek, tell, close에 해당하는 my\_create, my\_remove, my\_open, my\_file\_size, my\_seek, my\_tell, my\_close 함수를 추가한다. 또한 이를 제어할 syscall\_handler에 각 system call number에 따른 case 문을 추가해준다. Proj1에서 구현한 read, write에 해당하는 my\_read, my\_write 함수는 각각 fd가 0, 1인 경우에 대해서만 구현되어 있는데 fd가 3 이상인 file 입출력또한 추가한다. Read, write, open에서 synchronization을 구현하기 위해 syscall.c에 전역 변수로 lock 구조체 filesys\_lock을 선언하고 이를 이용해 critical section에 여러 process가 동시에 진입할 수 없도록 한다. 또한 실행 파일을 쓰고 제거하는 것을 방지하기 위해 filesys/file.c의 file\_deny\_write를 사용해야 하는데 이를 위해 해당 소스 파일에 정의된 file 구조체를 syscall.c에서 접근 가능하도록 userprog/syscall.h에 그대로 추가해준다.

1. **연구 결과**
   1. **Flow Chart**

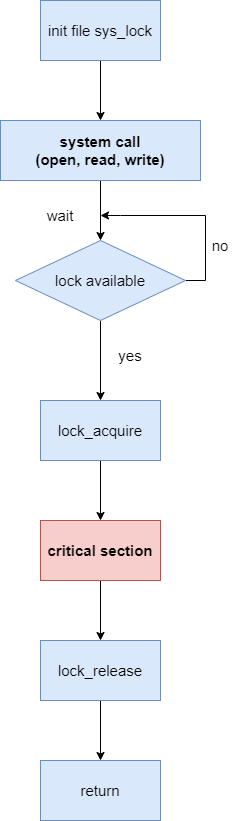
Thread 구조체에 선언한 file 포인터 배열 fd와 system call open으로 file descriptor 할당하는 과정



System call 호출은 Proj1과 큰 차이가 없다.



Lock을 얻어서 critical section에 진입하고 critical section 수행 후 lock을 반환하는 과정



* 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

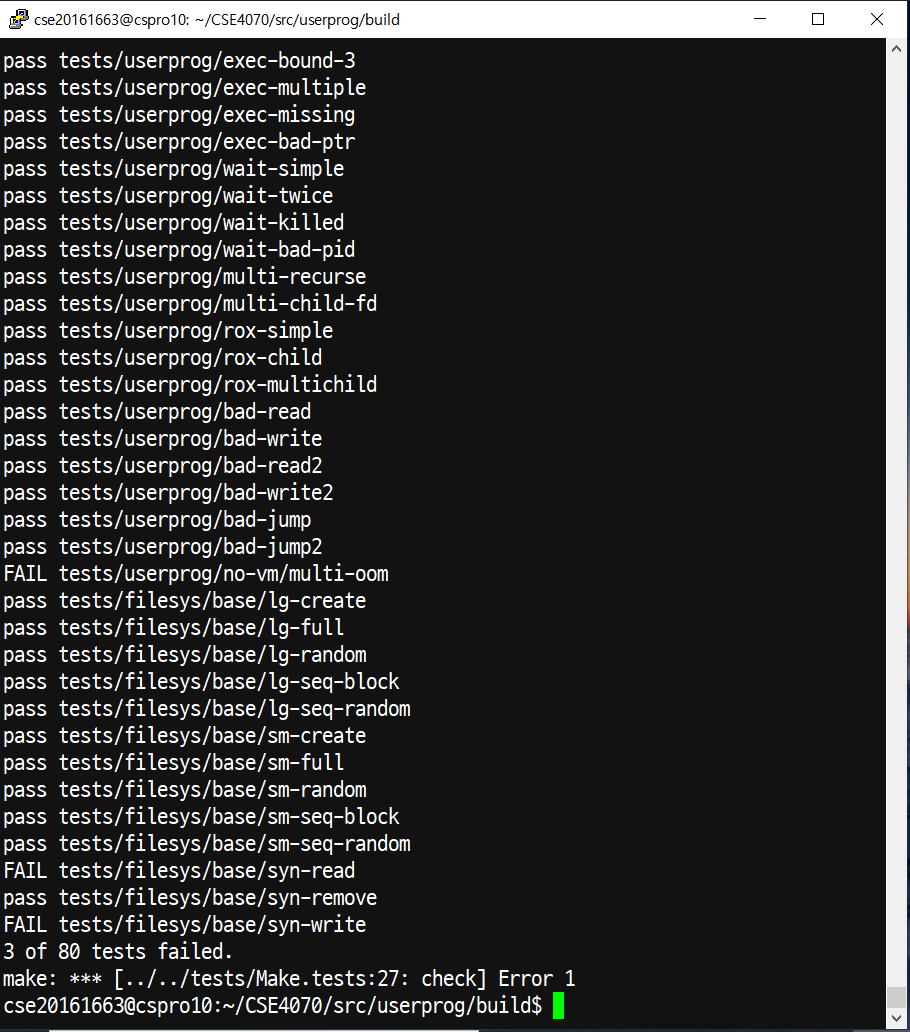
threads/thread.h의 thread 구조체에 struct file\* fd[128];을 선언하여 thread에서 file descriptor 번호에 따라 fd 배열에 저장된 file 포인터로 파일에 접근 가능하도록 하였다. init\_thread 함수에서 fd 배열의 모든 원소 값을 NULL로 설정 후, open으로 file을 열 때, fd 배열을 index 3(0, 1, 2 제외)부터 탐색해 아직 파일에 할당되지 않은 NULL인 인덱스를 찾아 해당 값을 file descriptor로 하도록 구현하였다.

System call은 Proj1과 마찬가지로 lib/syscall-nr.h의 system call 번호를 확인해 각 system call의 번호를 알아내고 이를 이용해 올바른 system call을 호출하도록 구현하였다. System call handler에 각 system call 구현함수인 my\_syscall(ex. my\_open, my\_close 등등)을 추가하였다. 대부분의 추가 system call은 filesys/filesys.c의 함수를 호출하거나 filesys/file.c의 함수를 호출하는 것으로 구현이 가능했다.

System call 구현 중 인자로 받는 file 문자열이 NULL일 경우, my\_ 함수에서 Proj1에서 구현한 my\_exit(-1)을 호출해 종료시키는 것을 구현하였다. 또한 인자로 주어진 fd에 해당하는 file 포인터가 NULL일 경우 할당된 파일이 없으므로 이 경우에도 my\_exit(-1)을 호출해 종료시키는 것으로 예외를 처리했다.

Synchronization을 위해 threads/synch.h에 정의된 lock 구조체를 이용해, filesys\_lock을 선언하고 threads/synch.c에 정의된 lock\_acquire를 통해 lock을 얻어 critical section에 진입, critical section의 실행이 끝나고 lock\_release를 통해 lock을 반환하여 critical section의 보호를 구현하려고 시도했다.

close를 구현한 my\_close 함수에서 현재 thread의 fd에 해당하는 file 포인터를 file\_close 함수를 호출해 닫고 NULL로 설정해야 하는데 NULL로 설정 후 닫아서 항상 file\_close에 NULL이 전달되었었다. 이를 수정하여 rox-child, rox-multichild를 해결하였다.

* 1. **시험 및 평가 내용**
* 

multi-oom, syn-read, syn-write FAIL