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

담당 교수 / 분반 : 김영재 / 1분반

이름 / 학번 : 신지원 / 20211547

개발 기간 : 10/24 ~ 11/7

1. **개발 목표**

Project 1에서는 프로세스를 제어하는 작업을 해주었다면, project 2 에서는 파일을 조작하는 작업을 해주고자 한다. System call 을 호출하여 특정 file 에 따른 구체적인 작업을 할 수 있도록 추가 작업을 구현할 것이다. File Descriptor, System Calls, Synchronization in Filesystem 의 구현을 통해 여러 파일 작업을 수행하고 이를 저장할 것을 목표로 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

1. File Descriptor

기본적으로 할당되는 파일디스크럽터는 프로젝트에서 접근할 수 없기 때문에 0~2 를 제외한 3부터 접근하며 File Descriptor 을 구현해야 한다. 이를 통해 여러 file 작업을 수행하기 위한 입력을 구현할 수 있다.

2. System Calls

이번 프로젝트에서는 file 과 관련된 system Call 을 구현해야 한다.

file 을 생성하는 create, file 을 지우는 remove, file 을 여는 open, file 의 size 를 return 하는 filesize, file 을 찾는 seek, tell, 열려있는 file 을 닫는 close 을 구현해야 한다. 추가적으로 project 1에서 구현하였던 write, read 도 1 에서 구현한 File Descriptor 에 맞도록 수정해주어야 한다. 이를 통해 1에서 구현한 file descriptior 를 수행할 수 있게 될 것이다.

3. Synchronization in Filesystem

User 는 각 system call 을 호출할 때 각자의 threads 에서 호출하게 된다. 이는 하나의 handler 에서 처리해주며 kernal 에서는 각자 호출한 system call 을 수행하는 동시에 여러 threads 에서의 task 들을 동기화해주어야 한다. 따라서 project 1 에서 구현하였던 process 코드에 추가로 stnchronization 해주는 코드를 추가해야 한다. 따라서 kernal 은 여러 trheads 에서 작업한 task 들을 원활하게 동기화할 수 있을 것이다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

**1. File Descriptor**

File Descriptor Table 을 구현하기 위해서 배열이라는 자료구조를 사용하였다. 그 이유는 인덱스를 통해 파일을 접근하기 용이하기 때문이다. 안내받은 메뉴얼에서 최댓값을 128로 정의하고 있기 때문에 fd[128] 로 설정하였다.

**2. System Calls**

2-1. Create

Create 는 file 이 없을 때 file을 생성할 때 요청하는 system Call 이다. 따라서 인자로 이름과 사이즈를 받아, 파일을 생성 성공시 true, 실패시 false 을 return 한다.

2-2. Remove

Remove 는 file 을 삭제할 때 요청하는 system Call 이다. 따라서 지우고자 하는 파일을 입력받고, 삭제 성공시 true, 실패시 false 를 return 한다.

2-3. Open

Open 은 file 을 열고자 할 때 요청하는 system Call 이다. thread 의 우선권을 얻은 뒤 확인해주고 file 을 open 해주고 thread 의 우선권을 반환하는 작업이 필요하다. 따라서 성공적으로 open 되었을 때 file descriptior 값이 반환되며, 실패시 -1이 반환된다.

2-4. Filesize

Filesize 는 해당 파일의 사이즈를 알고자 할 때 요청하는 system call 이다. 성공시 file 의 length 값이 반환된다.

2-5. Seek

Seek 는 특정 파일을 찾고자 할 때 요청하는 system call 이다. 따라서 인자로 특정 파일과 위치를 요청한다. 따라서 파일의 위치를 찾을 수 있다.

2.6 Tell

Tell 은 특정 파일을 찾고자 할 때 요청하는 system call 이다. Seek 과의 차이점은 pointer 로offset 으로 위치를 표현하는 seek 과 달리 tell 은 pointer 로 위치만을 반환한다.

2.7 Close

Close 는 특정 파일을 닫고자 할 때 요청하는 system call 이다. Close 한 후에 file descriptior 의 값이 열리기 전과 같이 NULL 로 변경된다.

2.8 Read

Read 는 특정 파일을 읽고자 할 때 요청하는 system call 이다. 이미 열려 있는 파일이 있을 때 이에 대해 우선순위를 잡아주기 위하여 lock 을 구현해야 한다.

2.9 Write

write 는 특정 파일을 읽고자 할 때 요청하는 system call 이다. 이미 열려 있는 파일이 있을 때 이에 대해 우선순위를 잡아주기 위하여 lock 을 구현해야 한다.

**3. Synchronization in Filesystem**

동기화하기 위한 방법으로는 lock, semaphore 가 있다. Lock 은 한 thread 에서만 접근 가능하며 락을 실행했을 땐 꼭 해제해주어야 한다는 특징이 있다.

Semaphore 의 경우 여러 thread 에서 접근가능하며 꼭 해제해주지 않아도 된다는 차이점을 갖는다.

3-1. lock

여러 thread 를 처리하고 있기 때문에 현재 파일에 대해 안전하게 접근하기 위해선 다른 thread 를 멈출 필요가 있다. 따라서 lock 을 사용하면 다른 thread 는 현재 파일에 접근할 수 없도록 잠글 수 있다. lock\_ acquire 통해 락을 소유하고 lock\_release 을 통해 락을 해제할 수 있다. 이번 프로젝트에서는 syscall 과 process\_excute 를 동작시키기 위하여 lock 을 사용하였다.

3-2. semaphore

세마포어의 경우 한 번에 여러 스레드가 자원에 접근할 수 있다. 이는 세마포어의 초기 값으로 설정되는데 0일 때는 점유 상태를 의미하며, 사용하기 위해선 1이상이 될때까지 대기하여야 한다. 세마포어는 일반적으로 wait/signal 연산을 통해 자원 접근을 제어한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

10/24 ~ 10/25: 프로젝트 설계 및 매뉴얼 이해

10/26 ~ 10/31: File Descriptor & System Calls 구현

11/1 ~ 11/5: Synchronization in Filesystem 구현

11/5 ~ 11/7: 에러 해결 및 디버그

11/7: 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.h 에서 file descriptor 에 필요한 fd[128] 과 c\_flag 를 선언 한다. thread.h 에서 선언한 인자들을 thread.c 에서 초기화해주며 file descriptor 의 구현 준비를 한다.
* **2. System Calls**
* - sysCreate, sysRemove, sysOpen, sysFlesize, sysSeek, sysTell, sysClose 을 구현하기 위해서 userprog/syscall.h 에서 함수들을 선언해준 뒤, userprog/syscall.c 에서 구체적인 함수를 구현한다.
* - 지난 프로젝트에서 구현하였던, sysWrite, sysRead 에 대한 함수도 앞서 준비해놓은 file descriptor 을 활용하여 추가 구현한다.
* **3. Synchronization in Filesystem**
* - threads/thread.h 에서 file\_lock 구조체를 추가하고 threads/thread.c 에서 이를 초기화하는 코드를 구현하여 lock 을 사용할 수 있도록 준비한다.
* - userprog/syscall.c 에서 sysOpen, sysRead, sysWrite 함수에서 file\_lock 을 통해 thread 를 소유할 수 있도록 acquire() 를 구현하며, 추후 release() 또한 호출하여 Lock 을 구현한다.
* - userprog/process.c 에서 process\_execute, start\_process 함수에서는 여러 스레드에서의 접근을 제어하는 semaphore 을 사용하여 동기화를 구현하였다.

1. **연구 결과**
   1. **Flow Chart**

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

**1. File Descriptor**

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

**2. System Calls**

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**3. Synchronization in Filesystem**

**텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

* 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

텍스트, 스크린샷, 폰트이(가) 표시된 사진

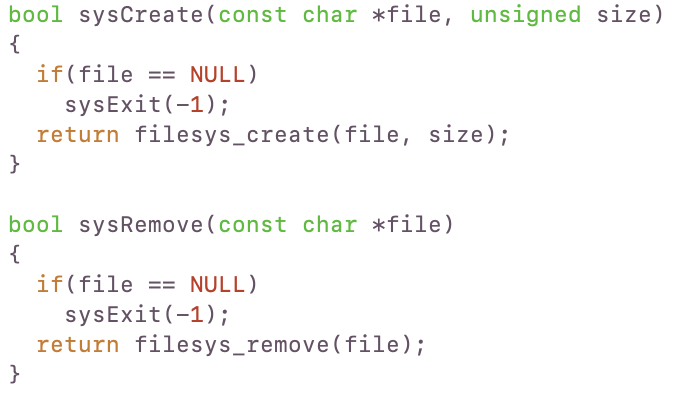
자동 생성된 설명텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명

Project 1에서 설정해놓았던 인자에 추가로 fd[128] 을 구현하였다. 앞서 설명하였던 것처럼 다른 자료구조보다 인덱스에 대한 접근이 용이한 배열로 구현하였다. c\_flag 는 현재 description 의 숫자를 담기위한 인자다. 이를 통해 현재 file 이 어떤 상태임을 저장할 수 있다. 따라서 thread.h 에서 인자들을 선언하고 thread.c 에서 이를 초기화 해주었다.

2. System Calls

2-1. Create & Remove



Create 와 remove 모두 파일에 대한 로직을 처리하기 때문에, 인자로 받은 file 이 존재하는 파일이어야 한다. 따라서 file 에 대한 NULL 검사를 해주어 만약 빈 파일일 떄는 종료처리한다. 만약 그렇지 않을 때는 각각의 file 함수로 처리하여 결과값을 true 혹은 false 로 반환한다.

2-2. Open

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

Open 은 파일에 대한 로직을 처리하기 때문에, 인자로 받은 file 이 존재하는 파일이어야 한다. 따라서 file 에 대한 NULL 검사를 해주어 만약 빈 파일일 떄는 종료처리한다. 유효한 파일이 경우, 파일 접근을 위해 lock 을 실행해 주었으며 자세한 것은 3번에서 설명하도록 하겠다. 에러 처리를 통과한 경우엔 file 함수를 처리하는 최솟값부터 최댓값까지 for 문을 돌면서 비어있는 공간을 찾는다. 만약 비어있다면 이곳에 file 포인터를 저장하도록 한다.

2-3. Filesize & Seek & Tell

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

filesize, seek, tell 모두 현재 스레드에서 열린 파일에 대해 로직을 처리하기 때문에 이에 대한 에외처리가 필요하다. 따라서 thread\_current() 에서 현재 파일이 NULL 인지 확인해주고 그럴 땐 강제 종료하였다. 만약 그렇지 않다면, 각각에 대해 결과를 출력해주었다.

Filesize 에 대해서는 현재 파일에 대해 길이를 출력해주는 함수를, seek 는 현재 파일의 offset 위치를, tell 의 경우에는 file\_tell 을 반환하며 이는 pointer 에 대한 값을 반환하는 함수다.

2.4 Close

텍스트, 폰트, 스크린샷, 대수학이(가) 표시된 사진

자동 생성된 설명

Close 도 마찬가지로 열려있는 파일을 닫는 과정이기 때문에 파일이 없다면 종료처리한다. 만약 유효한 파일이라면, file\_close 함수를 호출하여 현재 스레드가 가리키고 있는 file description 을 전달하고 해당 부분을 다른 파일이 사용할 수 있도록 전달된 file description 을 NULL 처리 해준다.

2.5 Read & Write

Read 와 Write 는 project1 에서 진행했던 코드에서 추가적으로 구현하였다. 기존에는 모든 파일을 구분없이 처리했다면, 이번 프로젝트에서는 file 함수를 사용하여 처리하여야 되기 때문에 길이에 대한 분기처리가 필요하다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

Read 는 파일이 0일 때, Write 는 file description이 1일 때면, project 1에서 구현한 그 방식으로, 3이상이라면 filesys.h 에 구현되어 있는 file 함수들을 호출하여 이를 처리해주었다. file 이 없다면 프로그램이 종료되도록, file 이 있다면 file\_read 혹은 file\_write 함수를 호출하여 처리하도록 구현하였다.

**3. Synchronization in Filesystem**

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명**

Lock 을 구현해주기 위하여 thread.h 에서 file\_lock 을 선언해주고, thread.c 에서 이를 초기화해주었다. 따라서 thread 가 초기화될 때 파일을 안전하게 접근하도록 lock 을 거는 상태가 되었다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

sysCall 중 Open, Read, Write 에서 file descriptor 에서 초기화하였던 file\_lock 을 사용하였다. 이는 파일 작업을 위해 thread 를 점유하여 안전한 작업을 위하여 위와 같이 구현하였다. 파일에 대한 작업을 모두 마친 뒤엔 lock\_release 를 하여 점유권을 해제하도록 하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

그 뒤 userprog/process.c 내부 함수들에서 semaphore 를 이용하여 스레드를 관리하였다. 위 코드는 process\_execute() 함수이다. 가장 먼저 현재 실행 중인 thread 를 가져오고 자식 스레드 포인터를 초기화한다. 그 뒤 자식 스레드 리스트를 순회하면서 유효한 자식 스레드를 찾는다. 그리고 그 자식 스레드가 작업을 완료할 때까지 현재 스레드가 대기할 수 있도록 구현하였다.

텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명

그 뒤 start\_process() 함수에서 부모 스레드의 대기를 해제한다. 이를 위하여 원래 구현되어있던 success 인자를 활용하여 현재 thread 의 대기 해제 여부를 판단하여 주었다. 그 뒤 sema\_up 함수를 통해 부모 스레드에게 작업이 완료되었음을 알리고 대기를 해제하도록 한다.

* 1. **시험 및 평가 내용**

텍스트, 스크린샷, 패턴이(가) 표시된 사진

자동 생성된 설명