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

담당 교수 / 분반 : 김영재 교수님/2반

이름 / 학번 : 김희진/ 20211530

개발 기간 : 10.20~10.28

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

File descriptor를 구현하고 이와 pintos에 내장된 함수를 이용하여 File system과 관련된 system call인 create, remove, open, close, filesize, read, write, seek, tell을 구현한다. 또한 이 과정에서 발생하는 synchronization을 semaphore, lock등을 이용해 해결한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목을 구현해야 하는 이유, 혹은 구현 시 기대되는 결과를 간략히 서술

1. File Descriptor  
각 Process는 file descriptor로 이루어진 array를 가지며, file을 열 때마다 file descriptor 하나를 할당해 file을 관리한다. 그렇기에 file system과 관련된 함수를 구현하기 위해서 file descriptor가 필요하다. 0과 1, 2는 STDIN, STDOUT, STDERR로 고정되어 있기에 우리는 fd 3번부터 이용 가능하다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

앞서 언급한 file descriptor를 이용해 File system과 관련된 system call인 create, remove, open, close, filesize, read, write, seek, tell을 구현한다. Project 1에서 stdin, stdout에 대해 작동하도록 구현했던 read와 write는 기능을 확장시켜 인자로 받은 file에 대해서 읽고 쓸 수 있도록 해야 한다.

3. Synchronization in Filesystem

Multi thread에서 file과 process에 대해 synchronization을 수행해야 한다. 하나의 파일에 대해 동시에 여러 thread가 접근하는 것을 막아 파일의 일관성을 유지해야 하며 parent나 child의 상태를 확인하여 orphan process나 zombie process가 생기는 것을 막아야 한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

Filesys/file.c에 정의되어 있는 struct file을 이용했다. 핀토스 메뉴얼에 file descriptor array의 크기는 128이라고 되어있기에 struct file \*fd[128]을 thread 구조체에 추가하여 사용했다. File 구조체에는 file에 대한 정보를 담고 있는 inode 구조체 포인터와 position을 담고 있는 element 그리고 file\_deny\_write() 함수의 call여부를 담고 있는 element가 있기에 이 자료구조를 사용해주었다.

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)

1) create

인자로 받은 file의 이름을 가지는 새로운 file을 create한다. 성공적으로 create했다면 1을 아니라면 0을 반환하고 file이 NULL일 때 예외처리도 해준다.

2) remove

인자로 받은 file의 이름을 가진 file을 remove한다. 성공적으로 remove했다면 1을 아니라면 0을 반환하고 file이 NULL일 때 예외처리도 해준다.

3) open

인자로 받은 file이 이름을 가진 file을 open한다. File이 없다면(null) -1을 반한하고 있다면 open한 file이 할당된 file descriptor의 번호를 반환한다. 이때, 같은 file을 여러 번 열면 열 때 마다 새로운 fd가 할당된다.

4) read

Stdin에 대한 read의 경우 input에 대해 읽고, 아닌 경우 fd에 해당하는 file에 대해 buffer로 읽어준 후 읽은 문자열 길이를 반환한다. 후자의 경우 fd의 해당하는 값이 NULL일 때 예외처리도 필요하다.

5) write

Stdout에 대한 write의 경우 콘솔창에 쓰고, 아닌 경우 fd에 해당하는 file의 buffer에 size bytes만큼 쓰고 써준 문자열 길이를 반환한다. 후자의 경우 fd의 해당하는 값이 NULL일 때 예외처리도 필요하다.

6) filesize

인자로 받은 fd에 해당하는 file의 length를 byte 단위로 반환한다. 이때 인자로 받은 fd에 파일이 할당되어 있지 않을 때 예외처리 또한 해준다.

7) close

인자로 받은 fd에 파일이 할당되어 있지 않을 때 예외처리를 해준다. 그렇지 않은 경우, 해당 fd에 해당하는 file을 close한다. 또한, fd에 해당하는 값도 NULL로 바꿔 파일이 할당되어있지 않은 상태로 한다.

8) seek

인자로 받은 file descriptor에 해당하는 file에서 read나 write할 위치를 인자로 받은 position으로 바꿔준다.

9) tell

인자로 받은 file descriptor에 해당하는 file의 read나 write할 위치를 return한다.

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

File에 동시에 2개 이상의 thread가 접근하는 것을 막기 위해 lock 구조체 file\_lock을 선언해 사용했다. Lock\_acquire함수를 이용해 lock시켜주고 lock\_release함수를 이용해 unlock 해주었다. 이는 file에 대해 접근하는 read, write, open system call에서 필요하며 위 두 함수로 file 접근에 대한 명령어을 감싸 주는 방식을 이용했다.  
또한, file에 여러 writer가 접근하는 것을 막기 위해 file\_deny\_write()를 사용했다.

또한, parent가 child보다 먼저 끝나는 것 혹은 child가 끝난 지 모르는 상황을 방지하기 위해 semaphore s\_load와 child의 load실패 여부를 나타내는 flag, 그리고 wait 함수를 사용했다. s\_load는 child가 load 하기 전 parent가 끝나는 것을 방지하기 위해서 이용했고 이 과정에서 sema\_up, sema\_down 함수를 사용해 lock, unlock하는 방식을 사용했다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

10.20-22 : 프로젝트 및 핀토스 매뉴얼 파악

10.23 : file descriptor 구현 및 system call 구현

10.27 : synchronization 구현

10.28 : 에러 해결 및 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.h의 struct thread에 file 구조체 포인터 fd[128]을 추가해 준 후, thread.c의 init\_thread()에서 초기화 하는 코드를 추가한다.

- userprog/syscall.c에서 fd가 필요한 즉, file system관련 system call을 수행하는 함수에서 이를 사용한다.  
null인 fd에 인자로 받은 file을 할당해주는 syscall\_open()과 인자로 fd를 받는, syscall\_fiesize(), syscall\_read(), syscall\_write(), syscall\_seek(), syscall\_tell(), syscall\_close()에 관련 코드를 추가했다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

- userprog/syscall.c에서 이번 프로젝트에서 구현해야 하는 system calls를 수행하는 함수들을 추가한 후, 이를 userprog/syscall.h에도 추가해준다.   
Syscall\_create, syscall\_remove, syscall\_open, syscall\_filesize, syscall\_seek, syscall\_tell, syscall\_close이다.

- userprog/syscall.c에서 저번 프로젝트에서 구현한 함수인 syscall\_write, syscall\_read에 대해서 stdout, stdin뿐만 아니라 file에 대한 write, read도 수행할 수 있게 한다.

3. Synchronization in Filesystem

- threads/thread.h에서 lock 구조체 file\_lock을 추가한 후, threads/thread.c의 thread\_init()에서 초기화하는 코드를 추가한다.  
file을 열고 읽고 쓰는 것과 관련된 system call인 syscall\_open, read,   
write(userprog/syscall.c)에서 file\_lock을 사용해 executable file에 다른 thread가 접근하는 것을 막는, file을 보호하는 코드를 추가한다.

* threads/thread.h의 struct thread에 semaphore s\_load와 int c\_flag, struct thread \*parent, struct file \*cur\_file를 추가해 준 후, thread.c의 init\_thread()에서 초기화하는 코드를 추가한다.  
  zombie process나 orphan process의 생성을 막기 위해서, userprog/process.c의 process\_execute, start\_process, load함수에서 이를 사용하는 코드를 추가한다. 이때, sema\_up, sema\_down 함수를 이용해 lock, unlock하는 방식을 이용했다.

1. **연구 결과**
   1. **Flow Chart**

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

1. File Descriptor

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

도표, 스케치, 텍스트, 그림이(가) 표시된 사진

자동 생성된 설명2. (이번 프로젝트에서 구현해야 하는) System Calls

3. Synchronization in Filesystem

-file lock

**텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 도표, 원이(가) 표시된 사진

자동 생성된 설명**

-semaphore, c\_flag, parent

**텍스트, 스크린샷, 원, 폰트이(가) 표시된 사진

자동 생성된 설명**

**도표, 평면도, 스케치, 기술 도면이(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

- threads/thread.h

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Struct thread에 file struct pointer fd[128]을 선언해 file descriptor로 사용했다. File descriptor의 크기 128은 핀토스 매뉴얼을 참고했고 file struct는 filesys/file.c에 있는 구조체이다.

- threads/thread.c



Init\_thread()에서 file descriptor를 NULL로 초기화시킨다.

Fd는 filesystem 관련 system call에서 사용되는데 이는 아래에서 자세히 설명했다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

-userprog/syscall.c

1. syscall\_create

인자로 받은 file이 NULL일 때 예외처리를 해주고 filesys\_create함수를 이용해 인자로 받은 file이름의 file을 만든다. Filesys\_create는 filesys/filesys.c에 내장되어 있는 함수로 create 성공 여부를 return한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

2. syscall\_remove

인자로 받은 file이 NULL일 때 예외처리를 해주고 filesys\_remove함수를 이용해 인자로 받은 file이름의 file을 제거한다. Filesys\_remove는 filesys/filesys.c에 내장되어 있는 함수로 remove 성공 여부를 return한다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

3. syscall\_open

인자로 받은 file이 NULL일 때 예외처리를 해주고 인자로 받은 file이름의 file을 filesys\_open을 통해 열어준다. Filesys\_open은 filesys/filesys.c에 있는 함수로 filesys/file.c에 있는 file\_open함수를 call해서 struct file을 return 한다. Filesys\_open을 통해 연 파일이 NULL이라면 -1을 return하고 이가 아닐경우 3부터 127까지 for문을 돌며 비어있는 file descriptor를 찾아 연 file을 fd에 할당한다. 그리고 해당 file descriptor의 번호를 반환한다. for문을 돌았음에도 비어있는 fd를 찾지 못한 경우에는 -1을 반환한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

4. syscall\_filesize

인자로 받은 번호에 해당하는 fd에 할당하는 파일의 크기를 file\_length함수를 이용해 반환한다. File\_length는 filesys/file.c에 있는 함수이며 filesys/inode.c에 있는 inode\_length를 call해서 file의 크기를 반환한다. 이때 해당하는 fd에 아무것도 없다면 예외처리를 해준다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

5. syscall\_read

지난 프로젝트에는 fd가 0일 때, 즉 stdin에 대한 read만 수행해주었다면 이번 프로젝트에서는 이외의 경우 read를 구현해줄 것이다. 그렇기에 fd가 0인 경우의 설명은 생략한다. 이를 위해 지난 프로젝트에서 구현했던 check\_address()를 이용해 buffer가 user 영역에 있는지를 check한다.

이후 fd가 3이상일 때 filesys/file.c에 있는 file\_read함수를 이용해 인자로 받은 fd에 해당하는 파일에 대해 buffer로 읽어준다. File\_read함수는 읽은 byte를 반환한다. 이때 인자로 받은 fd가 비었다면 예외처리를 해준다. 또한 fd가 3이상도 0도 아닌 경우에 대해서도 예외처리를 해준다.

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

6. syscall\_write

지난 프로젝트에는 fd가 1일 때, 즉 stdout에 대한 write만 수행해주었다면 이번 프로젝트에서는 이외의 경우 write를 구현해줄 것이다. 그렇기에 fd가 1인 경우의 설명은 생략한다. 이를 위해 지난 프로젝트에서 구현했던 check\_address()를 이용해 buffer가 user 영역에 있는지를 check한다.

이후 fd가 3이상일 때 filesys/file.c에 있는 file\_write함수를 이용해 인자로 받은 fd에 해당하는 파일에 대해 buffer로 읽어준다. File\_write함수는 읽은 byte를 return한다. 이때 인자로 받은 fd가 비었다면 예외처리를 해준다. 또한 fd가 3이상도 1도 아닌 경우에 대해서도 예외처리를 해준다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

7. syscall\_seek

인자로 받은 index의 fd가 비었다면 예외처리를 한다. filesys/file.c에 있는 file\_seek()을 이용해 파일의 read나 write를 수행하는 위치를 인자로 받은 position으로 바꾼다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

8. syscall\_tell

인자로 받은 index의 fd가 비었다면 예외처리를 한다. filesys/file.c에 있는 file\_tell()을 이용해 파일의 현재 위치를 반환한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

9. syscall\_close

인자로 받은 index의 fd가 비었다면 예외처리를 한다. Filesys/file.c에 있는 file\_close()를 이용해 파일을 닫아주고 fd의 값을 NULL로 바꿔주어 이후에 열리는 새로운 파일이 할당될 수 있도록 한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Make check를 했을 때 없는 file에 접근한다는 에러표시가 떠서 userprog/exception.c에 file이 존재하지 않을 때 예외처리하는 코드를 추가했다.



3. Synchronization in Filesystem

-threads/thread.h

Struct lock file\_lock을 선언해서 사용한다. Lock 구조체는 threads/synch.h에 있으며 이 헤더파일은 include되어있다.



-threads/thread.c



Thread\_init()에서 file\_lock을 lock\_init함수를 이용해 초기화해 주었다. Lock\_init함수는 마찬가지로 threads/synch.h에있다.

-userprog/syscall.c

3가지 system call에서 lock\_release함수와 lock\_acquire함수를 이용하여 file\_lock을 사용했다. 한 file에 여러 thread가 동시에 접근하여 file이 훼손되는 것을 막기 위한 것이며 두 함수는 모두 threads/synch.h에 있는 함수이다.

File\_lock을 사용한 함수는 우선 syscall\_open이 있다. filesys\_open을 하기 전 lock\_acquire함수를 통해 lock해주었다. 연 file이 NULL인 경우, for문을 돌며 null인 fd를 찾아 file을 할당해 주는 경우 그리고 for문에서 null인 fd를 찾지 못한 경우 모두 unlock될 수 있도록 lock\_release함수를 3군데에 사용했다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Syscall\_read()에도 file\_lock이 사용되었다. if문을 시작하기 전 lock\_acquire를 call해 lock해주고 fd==0인 경우, fd가 3이상인 경우 그리고 그 외의 경우 모두 unlock이 될 수 있도록 if, else if, else 3군데에 lock\_release를 call해주었다.

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

Syscall\_write()에도 file\_lock이 사용되었다. if문을 시작하기 전 lock\_acquire를 call해 lock해주고 fd==1인 경우, fd가 3이상인 경우 그리고 그 외의 경우 모두 unlock이 될 수 있도록 if, else if, else 3군데에 lock\_release를 call해주었다.

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

-threads/thread.h

Parent process와 child process의 synchronization문제를 해결하기 위해서 struct thread에 load를 위한 semaphore s\_load와 load에 실패한 child가 있음을 알리는 c\_flag, 그리고 parent process를 나타내는 thread \*parent 그리고 현재 쓰고 있는 파일을 나타내는 cur file을 선언해주었다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

-threads/thread.c

Init\_thread에서 위 변수들을 초기화한다. S\_load와 c\_flag는 0으로 그리고 parent는 현재 진행되고 있는 thread로(running\_thread()를 이용) 초기화 해준다. 또한, cur\_file은 NULL로 초기화 해준다.

**텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명**

-userprog/process.c

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명Process\_execute()에서 thread\_create를 call하여 start\_rpocess()가 실행된 후 sema\_down(s\_load)를 call해 block시켜준다. 이 sema\_down으로 인해 해당 함수의 아랫줄은 더 이상 실행되지 않은 상태로 start\_process()가 계속 실행되게 된다.

위 함수(process\_execute)에서 start\_process()가 실행되고 start\_process()에서는 load()가 실행된다. Load가 성공적으로 실행되었을 때 cur\_file에 현재 load된 file을 담고 file\_deny\_write를 call해 write를 막아준다. 이는 filesys/file.c에서 제공되는 함수이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

load가 끝난 후에 load를 call했던 start\_process()에서는 sema\_up(parent->s\_load)를 통해 process\_excute에서 block되어 있던 부분을 unlock해준다. 그리고 child load가 성공적으로 이루어지지 않았을 때 c\_flag에 1을 넣어 이를 기억한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명Sema\_up을 통해 unblock 시켜주므로 process\_execute()에서는 sema\_down code의 아랫줄이 실행된다. For 문에서 list를 돌며 thread의 c\_flag가 1인경우, 즉 child가 정상적으로 load되지 않은 경우에는 process\_wait()을 call해 zombie process가 발생하지 않게 한다.

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

텍스트, 스크린샷, 메뉴, 패턴이(가) 표시된 사진

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

자동 생성된 설명텍스트, 스크린샷, 메뉴, 패턴이(가) 표시된 사진

자동 생성된 설명