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

담당 교수 / 분반 : 김영재 / 1반

이름 / 학번 : 유경연 / 20171659

개발 기간 : 10/3~10/10

1. **개발 목표**

File system과 관련한 system call을 구현해서 project 1에서 통과하지 못한 나머지 pintos test를 모두 통과할 수 있도록 한다..

1. **개발 범위 및 내용**
   1. **개발 범위**

**1. File Descriptor**

stdin, stdout 뿐만 아니라 임의의 file을 읽거나 쓰기 위해서 필요하다.

**2. (이번 프로젝트에서 구현해야 하는) System Calls**

이번 프로젝트에서 구현할 System Call들은 다음과 같다. create(), remove(), open(), filesize(), seek(), tell(), close(), read(), write().

**3. Synchronization in Filesystem**

여러 프로세스가 한 파일에 동시에 접근할 수 있으면 파일의 내용을 읽거나 쓰는 경우에 잘못된 값을 읽어올 확률이 높다. 따라서 한 파일을 동시에 접근할 수 없도록 제한해 파일의 값이 달라지는 것을 막아 잘못된 값을 읽어오지 않도록 한다.

* 1. **개발 내용**

**1. File Descriptor:**

각 thread마다 고유의 file descriptor를 보유해야 하므로 thread 구조체에 file descriptor를 추가한다.

**2. System Calls:**

* + - **bool create(const char \*file, unsigned initial\_size)** : 새로운 initial\_size byte의 크기를 가진 file을 생성한다. 성공적으로 생성하면 true를 반환하고 아닌 경우엔 false를 반환한다.
    - **bool remove(const char \*file)** : file을 삭제한다. 성공적으로 삭제하면 true를 반환하고 아닌 경우에 false를 반환한다.
    - **int open(const char \*file)** : file을 열고 음이 아닌 정수 file descripter를 반환한다. 이때 열지 못했을 경우에 -1을 반환한다.
    - **int filesize(int fd)** : file의 크기를 반환한다.
    - **void seek(int fd, unsigned position)** : file에서 읽거나 쓸 다음 byte의 위치를 바꾼다.
* **unsigned tell(int fd)** : file에서 읽거나 쓸 다음 byte의 위치를 반환한다.
* **void close(int fd)** : file desriptor를 닫는다.
* **int read(int fd, void \*buffer, unsigned size)** : fd로 열린 file에서 byte를 읽는다. 읽은 byte의 수를 반환하거나 읽을 수 없다면 -1을 반환한다.
* **int write(int fd, const void \*buffer, unsigned size)** : fd로 열린 file에 byte를 쓴다. 쓴 byte의 수를 반환한다.

**3. Synchronization in Filesystem**:

File의 Synchronization을 만족시키기 위해서 어느 한 프로세스가 file에 접근할 때 lock\_aquire()함수를 사용하고 file에서 빠져나올 때는 lock\_release()함수를 사용하도록 하였다. 이 방법을 통해 1개의 프로세스만 file system을 사용할 수 있다.

또한 thread간의 동기화가 필요한 경우엔 Semaphore를 사용해야한다. 이번 프로젝트에서는 자식 프로세스의 load가 끝나기 전에 부모 프로세스가 종료되는 것을 막기 위한Semaphore를 추가해주었다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**
   * 10/4 : System Calls 구현
   * 10/6 : File system Synchronization 구현
   * 10/7 : Semaphore 추가
   * 10/9 : multi-oom test pass
   1. **개발 방법**

**1. File Descriptor:**

수정할 소스코드: threads/thread.h

Thread 구조체에 file descriptor를 추가한다.

**2. System Calls**

수정할 소스코드: userprog/syscall.c

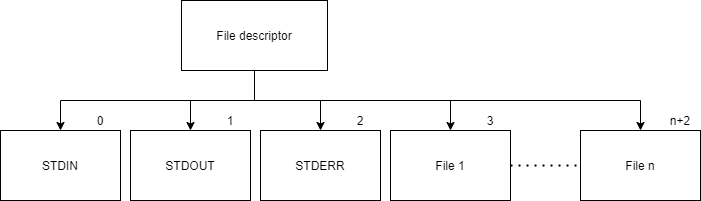
이번 프로젝트에서 구현할 system call 함수들을 추가한다.

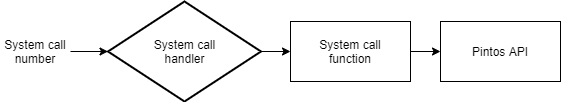
**3. Synchronization in Filesystem**:

수정할 소스코드: userprog/syscall.c, userprog/process.c, threads/thread.h

File system을 이용하는 함수들에 동기화를 위하여 lock을 걸고 thread간에 동기화를 위해 thread 구조체에 semaphore를 추가했다.

1. **연구 결과**
   1. **Flow Chart**

**1. File Descriptor**

**2. System Calls**

**3. Synchronization in Filesystem**

* 1. **제작 내용**

**1. File Descriptor**

각 thread마다 고유의 file descriptor를 보유해야 하므로 thread.h에서 thread 구조체에 struct file\* fd[128]을 추가하고 init\_thread()함수에서 NULL로 초기화하였다. 이때 배열의 크기는 pintos manual을 참고하였다. 또한 사용 중인 file descriptor의 개수를 저장하는 fd\_num 변수를 만들어서 open()시에 조금 더 용이하게 하였다. File을 사용하기 위해서는 fd에 NULL이 아닌 file\* 값을 저장하면 된다.

**2. System Calls**

프로젝트1에서 작성한 syscall.c에 추가로 작성하였다. 이번 프로젝트에서 구현해야 하는 system call 함수를 만들고 syscall\_handler()의 switch문에서 호출해주었다**.**

* + - **bool create(const char \*file, unsigned initial\_size)** : file이 만약 NULL이라면 exit()함수를 호출해서 종료되도록 하고 만약 NULL이 아니라면 pintos API인 filesys/ filesys.h/filesys\_create()를 호출하여 file을 생성했다.
    - **bool remove(const char \*file)** : file이 만약 NULL이라면 exit()함수를 호출해서 종료되도록 하고 만약 NULL이 아니라면 pintos API인 filesys/filesys.h/ filesys\_remove()를 호출하여 file을 삭제했다.
    - **int open(const char \*file)** : file이 만약 NULL이라면 exit()함수를 호출한다. NULL이 아니라면 pintos API인 filesys/filesys.h/filesys\_open()을 사용해서 file을 연다. File을 연 이후에 다른 프로세스에서 file을 수정하게 되면 값의 변화가 있을 수 있으므로 file을 열기전에 thread/thread.h/lock\_acquire()함수를 이용해서 lock을 걸어준다. 이렇게 하면 다른 프로세스에서는 lock이 release되기 전까지 file에 접근할 수 없다. File을 여는데 실패했을 경우 lock을 release하고 -1을 반환한다.

File을 열면 file discriptor의 값을 반환해야 하므로 fd에서 값이 NULL인 fd\_num부터 128사이의 index를 찾는다. 이때 만약 현재 thread의 name과 file이 같다면executable of running thread에 write하는 것을 막기 위해 filesys/file.h/file\_deny\_write()를 호출한다. 이후 File desriptor의 file pointer를 저장하고 fd\_num의 크기를 1 키우고 lock\_release()를 호출하고 index를 반환한다.

* + - **int filesize(int fd)** : 현재 file descriptor의 값이 NULL일 경우 exit()을 호출하고 아닐 경우 pintos API인 filesys/file.h/file\_length()를 호출한다.
* **void seek(int fd, unsigned position)** : **unsigned tell(int fd)** : 현재 file descriptor의 값이 NULL일 경우 exit()을 호출하고 아닐 경우 pintos API인 filesys/file.h/file\_seek()을 호출한다.
* **unsigned tell(int fd)** 현재 file descriptor의 값이 NULL일 경우 exit()을 호출하고 아닐 경우 pintos API인 filesys/file.h/file\_tell()을 호출한다.
* **void close(int fd)** : 현재 file descriptor의 값이 NULL일 경우 exit()을 호출하고 아닐 경우 pintos API인 filesys/file.h/file\_close()을 호출한다. 이때 file descriptor의 값을 NULL로 만들어주어야 한다.
* **int read(int fd, void \*buffer, unsigned size)** : 프로젝트1에서 구현한 함수를 stdin뿐만 아니라 file도 읽을 수 있도록 수정했다. 이때 입력으로 들어오는 buffer가 올바르지 않은 주소가 들어오는 경우를 예외처리하기 위해 프로젝트1에서 사용한 check\_vaddr()을 사용하였다.

Open()과 마찬가지로 동기화를 해주기 위해서 lock\_acquire()함수를 호출해서 다른 프로세스가 file에 접근하지 못하도록 하였다. Fd가 0인 경우는 STDIN이므로 buffer에서 읽었고 fd가 2 이상인 경우 pintos API인 filesys/file.h/file\_read()함수를 호출해서 file을 읽었다. 그리고 함수가 종료되기 전에 lock\_release()함수를 호출해서 다른 프로세스들이 file에 접근할 수 있도록 하였다.

* **int write(int fd, const void \*buffer, unsigned size)** 프로젝트1에서 구현한 함수를 stdout뿐만 아니라 file에도 쓸 수 있도록 수정했다. 이때 동기화를 위해 lock\_acquire()함수를 추가하였다. Fd가 1인 경우 stdout이므로 이전과 똑같이 구현하였다. Fd > 2 인 경우에 filesys/file.h/file\_write()를 호출하였다. Read()와 마찬가지로 함수가 종료되기 전에 lock\_release()함수를 호출했다.

**3. Synchronization in Filesystem**

File system에서의 동기화를 위해서 lock을 사용한 것은 위에서 설명한 것과 같다.

자식 thread의 load()가 완료되기 전에 부모 thread가 종료되는 문제가 발생해서 thread.h에 동기화를 위한 semaphore인 sema\_sync를 추가하고 init\_thread()에서 원래 있던 semaphore들과 똑같이 초기화했다. 또한 부모 thread를 저장하기 위해 만든 변수인 parent는 현재 실행중인 thread로 초기화했다.

Process.c/process\_execute에서 thread\_create()가 호출된 이후에 sema\_down()을 호출해서 부모 thread가 자식 thread의 load()가 끝나기를 기다린다. 자식 thread의 load()가 끝나면 자식 thread의 start\_process()에서 부모 thread의 semaphore에 대해서 sema\_up()을 호출하여 부모 thread가 실행될 수 있도록 했다.

또한 load()에 실패하는 thread가 생겨서 자식 thread를 생성한 이후에 자식 thread 중에서 load()에 실패한 thread를 뺐다.

* 1. 텍스트이(가) 표시된 사진

     자동 생성된 설명**시험 및 평가 내용**