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

담당 교수 / 분반 : 김영재 / 02

이름 / 학번 : 김 민 / 20171608

개발 기간 : 10/03 ~ 10/16

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

**Create, remove, open, read, write 등 file과 관련된 system call을 구현한다.**

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 프로세스는 독자적인 file descriptor를 가진다고 매뉴얼에 명시되어 있으므로 threads/thread.h에 정의되어 있는 struct thread에 file descriptor를 추가하여 프로세스마다 독자적인 file system을 가질 수 있도록 한다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

Create: 파일을 생성할 수 있게 된다.

Remove: 파일을 제거할 수 있게 된다.

Open: 파일을 열어 접근할 수 있게 된다.

Filesize: 파일의 크기를 반환한다.

Read: 파일을 open하여 내용을 읽을 수 있게 된다.

Write: 파일을 open하여 파일에 쓸 수 있게 된다.

Seek: 현재 open된 파일의 cursor를 변경할 수 있게 된다.

Tell: 현재 open된 파일의 cursor의 위치를 반환한다.

Close: 현재 open된 파일을 닫는다.

3. Synchronization in Filesystem

이번 프로젝트에서는 메모리에 올라가 있는 실행 파일의 변경(이미 메모리에 올라가 있어 변경되더라도 문제가 없지만)을 방지해야 한다. 또한 추가로 하나의 프로세스에서 파일을 읽고 쓰는 critical section을 진행하는 중 다른 프로세스가 같은 파일에 접근하는 것을 막아야한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

Filesys/file.c에 정의되어 있는 struct file의 배열을 threads/thread.h에 정의된 struct thread의 멤버로 추가하여 int type의 file descriptor를 받아 파일에 접근할 수 있도록 한다. Pintos에서 제공하는 file system api를 보면 struct file\* type을 parameter로 받거나 int type의 fd를 받도록 구현되어 있기 때문이다.

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: 파일을 생성할 수 있게 된다.

Create: pintos에서 제공하는 filesys\_create()을 실행시킴으로 해당 기능을 구현한다.

Remove: pintos에서 제공하는 filesys\_remove()을 실행시킴으로 해당 기능을 구현한다.

Open: pintos에서 제공하는 filesys\_open()을 실행시킴으로 해당 기능을 구현한다. 메모리에 올라간 실행파일을 open하지 않기 위해 file\_deny\_write()를 이용한 예외처리를 제공한다.

Filesize: pintos에서 제공하는 file\_length()을 실행시킴으로 해당 기능을 구현한다.

Read: pintos에서 제공하는 file\_read()을 실행시킴으로 해당 기능을 구현한다. 하나의 프로세스가 file을 read하는 것은 critical section이므로 critical section을 지키기 위해 struct lock을 사용한다.

Write: pintos에서 제공하는 file\_write()을 실행시킴으로 해당 기능을 구현한다. 하나의 프로세스가 file을 write하는 것은 critical section이므로 critical section을 지키기 위해 struct lock을 사용한다.

Seek: pintos에서 제공하는 file\_seek()을 실행시킴으로 해당 기능을 구현한다.

Tell: pintos에서 제공하는 file\_tell()을 실행시킴으로 해당 기능을 구현한다.

Close: pintos에서 제공하는 file\_close()을 실행시킴으로 해당 기능을 구현한다. File을 close하며 해당 file descriptor를 NULL로 만들어준다.

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

Lock은 semaphore의 특수한 형태로 1개의 process가 제어권을 독점, 반환하는 형식이다. 따라서 file 접근의 critical section에서 1개의 process만이 제어권을 가져 다른 process가 해당 파일에 접근하는 것을 막음으로써 synchronization을 구현할 수 있다. 추가로 이번 프로젝트에서는 child process의 load 또한 synchronization의 대상이 되므로 이를 위한 semaphore가 추가로 필요하다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

­10/3 ~ 10/7: 명세서, 매뉴얼 숙지, 구현해야 하는 system call의 기본적인 형태 구현, file.c, filesys.c에 구현된 system call api 파악.

10/8 ~ 10/12: file descriptor 추가, 각 system call의 구체적인 구현, 예외처리 구현

10/13 ~ 10/16: threads/synch.c 파악, synchronization을 위한 logic 추가, pass하지 못하는 test case에 대한 보완

* 1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드

Syscall.c의 switch case문에 새로운 system call을 위한 코드를 추가

load에 실패한 child process를 제거하기 위한 코드 추가

child process의 load 성공 여부를 검사하는 코드 추가

exception.c에 page fault를 처리하기 위한 조건 추가

* + 수정하거나 추가해야 하는 자료구조

Syscall.c에 synchronization을 위한 전역변수 struct lock file\_lock을 추가

Syscall.c에 struct file의 원형을 추가

Struct thread에 struct file\* [128]을 추가(file descriptor)

Struct thread에 자신의 parent process를 저장하기 위한 struct thread\* parent 추가

Struct thread에 child process의 load를 기다리기 위한 struct semaphore load\_wait 추가

Struct thread에 child process의 load 성공 여부를 저장하기 위한 int load\_check 추가

* + 수정하거나 추가해야 하는 함수

Process\_execute()에서 생성된 child process의 load의 성공을 확인, 실패했다면 child list에서 제거하는 코드 추가

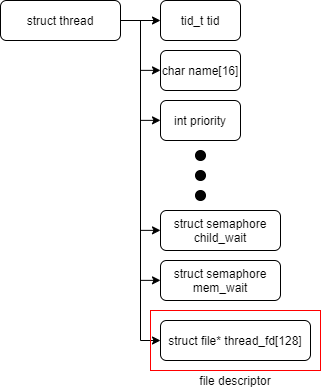
Start\_process()에서 child process의 load의 성공 여부를 기록, 실패했다면 thread\_exit()가 아닌 exit(-1)을 호출하도록 변경

Process\_exit()에서 process가 종료될 때 file descriptor를 전부 닫도록 코드 추가

1. **연구 결과**
   1. **Flow Chart**

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

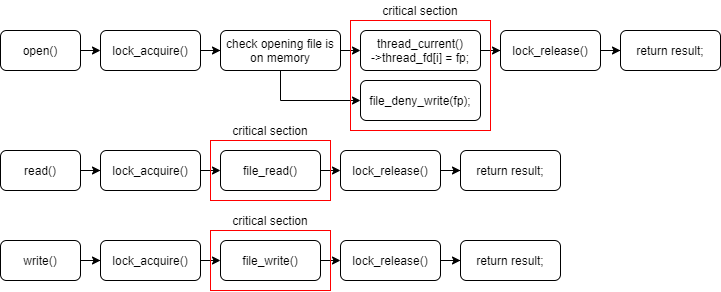
File descriptor



System call



Synchronization



* 1. **제작 내용**
* II. B. 개발 내용의 각 3가지 항목에 대하여 실직적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)

File descriptor의 경우 threads/thread.h에 정의된 struct thread에 filesys/file.c에 정의된 struct file의 포인터 배열을 멤버로 추가하는 것으로 구현하였다. System call은 thread\_current() 함수를 통해 file descriptor에 접근하여 file descriptor에 담긴 pointer를 이용해 file을 open한 뒤 read, write 등을 수행한다.

System call의 경우 create, remove, filesize, seek, tell은 file.c와 filesys.c에 이미 작성되어 있는 api를 호출하여 parameter를 전달하는 것만으로도 구현이 되었다.

Close는 file\_close()를 호출하기 전에 process의 file descriptor에 NULL을 넣어 해당 file descriptor를 비워주었다.

Read와 write의 경우 1번 프로젝트에서 구현한 내용에 추가로 파일 입출력을 구현하였다. 각각 해당하는 fd가 가리키는 file descriptor를 이용해 file\_read()와 file\_write()를 호출하였다. 그리고 후술할 synchronization에 대한 부분도 구현하였다.

Open의 경우 현재 process의 file descriptor에서 빈 곳을 찾아 그 곳에 filesys\_open을 이용해서 open한 file의 pointer를 저장한다. open에서는 후술할 critical section에 대한 protect와 running program에 대한 protect가 적용되어 있다.

Synchronization의 경우 open, read, write에 대해서 process가 파일에 접근할 때 다른 process가 접근하지 못하도록 protect 되어 있다. Critical section(file\_read(), file\_write(), filesys\_open()) 이전에 lock을 이용하여 lock\_acquire()를 호출함으로써 다른 process의 접근을 막고 critical section이 종료된 후에는 lock\_release()를 호출하여 제어권을 반환한다.

또한 open의 경우 현재 memory에 올라가 있는 실행 파일에 대해서 접근을 금지시켜야 하는데 이는 file\_deny\_write() 함수를 이용해 구현하였다. 만약 thread\_current()->name과 open하려는 file이 같을 경우 file\_deny\_write()를 호출하여 해당 파일에 접근하지 못하도록 구현하였다.

* 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명

System call을 구현하기 위해 file.c와 filesys.c에서 제공하는 api(filesys\_open(), file\_read(), file\_write(), file\_tell())등을 이용하였다.

Synchronization을 구현하기 위해 file.c에서 제공하는 file\_deny\_write()와 threads/synch.c에서 제공하는 lock\_acquire(), lock\_release()를 사용하였다.

* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명

Child process가 create 되어서 load되기 이전에 parent process가 종료되는 문제에 대해 고려하지 못해 통과하지 못하는 test case가 존재하는 것을 확인하였고, 이를 해결하기 위해 semaphore를 이용해 child process가 load될 때까지 wait하는 기능과 child process가 load 되었는지를 확인하고, 만약 load에 실패하였다면 child list에서 제거하는 logic을 추가로 구현하였다.

NULL을 open하거나 NULL로 jump하려는 test case에서 fail하는 것을 확인하고 system call을 호출하기 전에 전달받은 parameter가 유효한 parameter인지 확인하는 과정을 추가하였다.

일부 test case에서 page fault가 발생하는 경우 page\_fault에서 exit(-1)을 호출하지 못하는 것을 확인하고 검사 조건을 추가하여 문제를 해결하였다.

* 1. **시험 및 평가 내용**
* make check 수행 결과를 캡처하여 첨부
* 텍스트이(가) 표시된 사진

  자동 생성된 설명