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

담당 교수 / 분반 :

윤상현/20191613:

개발 기간 :

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

파일 디스크립터를 만들어서 유저가 파일에 입출력 할 수 있도록 합니다.

create, remove, open, close, filesize, read, write, seek, tell등 유저가 파일에 관여할 수 있게하는 시스템 콜을 구현합니다.

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 프로세스 별로 파일 디스크립터를 생성하여, 각 프로세스가 파일에 접근하거나 정보를 얻어 올 수 있게 합니다.

* + 1. (이번 프로젝트에서 구현해야 하는) System Calls

Read: 파일이나 stdin에서 정보를 읽어옵니다.

Write: 파일이나 stdout에 정보를 씁니다.

Create: 파일을 생성합니다.

Remove: 파일을 삭제합니다.

Open: 파일을 열어서 fp에 저장할 수 있게 합니다.

Close: 파일을 닫고 fp에서 삭제합니다.

Filesize: 파일의 크기를 알려줍니다.

Seek: 파일의 offset(커서 ?)를 이동합니다.

Tell: 파일의 offset을 알려줍니다.

이러한 일들을 유저가 할 수 있도록 시스템콜을 작성해야 합니다.

* + 1. Synchronization in Filesystem

한 프로세스가 특정 파일에 정보를 쓰거나 읽고 있을 때에, 다른 프로세스에서 똑같이 접근하여 정보를 변경하면 예상치 못한 작동이 일어날 수 있으므로, lock과 semaphore을 이용하여, 단일 프로세스만 접근하도록 합니다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술
  + 1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

배열을 사용할 것입니다. 특정 번호의 파일에 즉시 접근할 수 있고, 별도의 free과정이 필요하지 않기 때문입니다.

* + 1. 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: filesys\_create를 호출합니다. 파일을 만듭니다.

Remove: filesys\_remove를 호출합니다. 파일을 삭제합니다.

Open: filesys\_open으로 파일을 열고, 현재 프로세스의 fp에 해당 파일을 추가합니다. 현재 프로세스의 실행파일을 덮어쓰지 않기 위해서, 파일 이름이 현재 프로세스와같으면, file\_deny\_write를 실행시켜줍니다.

Close: filesys\_close로 파일을 닫고, 현재 프로세스의 fp에서 해당 파일을 삭제합니다.

Filesize: file\_length를 호출합니다. 파일의 크기를 알려줍니다.

Read: fd가 0이면 input\_getc로 stdin 입력받고, 아니면 해당 fp의 정보를 file\_read로 읽습니다.

Write: fd가 1이면 putbuf로 stdout 쓰고, 아니면 해당 fp에 file\_write로 씁니다.

seek: file\_seek을 호출합니다. 파일의 offset을 이동시킵니다.

tell: file\_tell을 호출합니다. 파일의 offset을 알려줍니다.

* + 1. Synchronization in Filesystem: Lock, Semaphore를 어떻게 이용할 수 있는지 각각에 대해 설명 (다른 방법을 서술해도 되지만 lock과 semaphore는 반드시 포함해야 함)

일단 한 프로세스가 파일에 접근할 경우, lock\_acquire을 걸어줍니다. 파일에 대한 여러 프로세스의 접근을 막기 위함입니다. ead, write의 함수 시작 부분에서 lock을 걸고, 파일과 관련된 처리가 끝난이후에 lock\_release로 lock을 풀어주어 다른 프로세스들도 접근할 수 있게 해줍니다. 또, 현재 프로세스의 실행파일이 훼손되는 것을 막기 위하여, 프로세스이름과 열려는 파일 이름이 같으면 file\_deny\_wirte를 호출해서 쓸 수 없게 합니다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

파일 디스크립터를 thread.h 헤더파일에 추가하고, 초기화 과정을 작성합니다.

Process.c에 파일 열고 닫기 관련 함수를 추가하고, process\_exit에 파일 전체 닫기 관련 구문을 추가합니다.

시스템콜을 syscall.c에 추가합니다.

그 과정에서 lock으로 synchronization을 구현합니다.

* 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

Thread.h, thread.c에서 thread 자료구조에 fp 배열 추가 및 thread\_create()에 초기화.구문 추가

Syscall.h에 새로운 시스템콜들 선언 및, synchronization을 위한 lock 선언

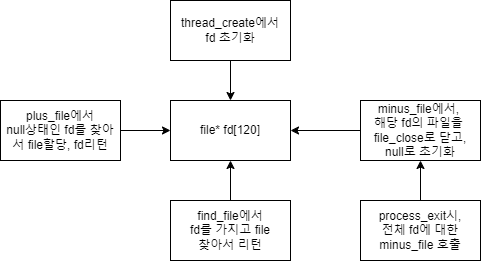
Syscall.c에서 새로운 시스템콜들 (create, remove, open, close, filesize, read, write, seek, tell )추가 및 syscall\_init()에 lock 초기화 구문 추가

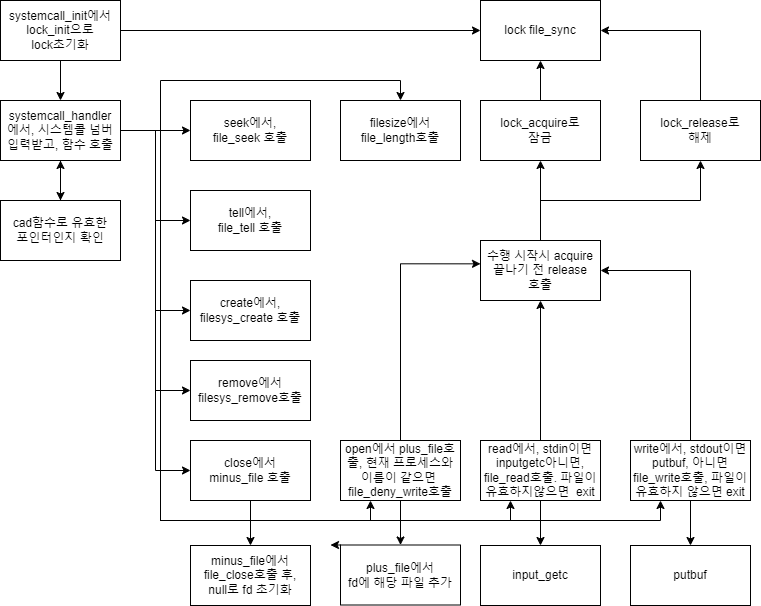
Process.c에 fd가 주어지면 해당 파일을 찾아주는 find\_file 함수, 파일을 fd에 추가하는 plus\_file 함수, 빼주는 minus\_file 함수를 만듭니다.

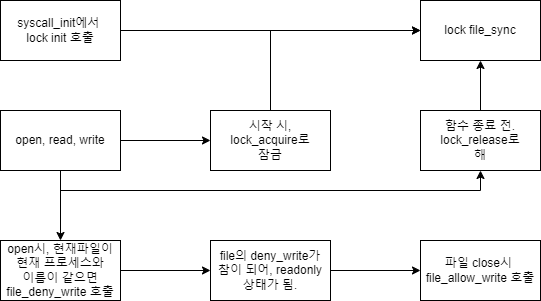
Process\_exit 함수에, 파일디스크립터의 파일을 닫는 구문추가.

1. **연구 결과**
   1. **Flow Chart**

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

****

****

****

* 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
* Thread.h, thread.c에서 thread 자료구조에 file\* [120] 크기의 정적 배열을 추가 및 thread\_create()에서 전부 null로 초기화 해주었습니다.
* Syscall.h에 새로운 시스템콜들 선언(create, remove, open, close, filesize, read, write, seek, tell ) 및, synchronization을 위한 lock\_ 선언하였습니다.
* Syscall.c에서 새로운 시스템콜들 추가하였습니다.
* Create : filesys\_create 호출
* Remove: filesys\_remove 호출
* Open: lock\_acquire을 호출해서 이 프로세스이외의 접근을 막고, plus\_file(다음 쪽에 설명)을 호출해서 fd에 추가해 줍니다. 그후 lock\_release로 lock을 해제합니다.
* close : minus\_file을 호출합니다. (다음 쪽에 설명)
* filesize,: file\_length 호출
* read,:fd가 이면 get\_inputc로 키보드의 입력을 받고, 아니면 해당 fd의 파일을 파일 디스크립터에서 가져와서 거기서 file\_read()호출후 읽은 바이트 리턴
* write: fd가 이면 put\_buf로 stdout에 출력해주고, 아니면 해당 fd의 파일을 파일 디스크립터에서 가져와서 거기서 file\_write()호출후 쓴 바이트 리턴
* seek: file\_seek 호출
* tell : file\_tell 호출
* syscall\_init()에 lock\_init으로 lock을 초기화해주었습니다.
* Process.c에
* find\_file: 파일 디스크립터에서 해당 fd번호의 파일을 찾아 리턴해줍니다. 파일이 할당 되어있지 않다면 null 을 리턴합니다.
* plus\_file: 비어있는 fd(null)을 찾아서 할당해주고, 해당 fd를 리턴해줍니다.
* minus\_file: 해당 fd의 파일을 file\_close를 호출하여 닫고, null로 초기화해줍니다.
* Process\_exit 함수에서, 전체 fd를 검사하여 null이 아닌 파일에 대하여 전부 minus\_file 을 호출하여 닫아주었습니다.
  1. **시험 및 평가 내용**
* make check 수행 결과를 캡처하여 첨부
* 텍스트이(가) 표시된 사진

  자동 생성된 설명