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

담당 교수 / 분반 : 김영재 / 2반

이름 / 학번 : 우석민 / 20190408

개발 기간 : 2023.10.25~2023.10.28

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

파일과 관련된 system call을 저번 프로젝트에서 구현을 하지 않았다. 또한 read/write도 한정적으로만 구현을 했었다. 이번 프로젝트에서는 이런 system call들을 구현하고, 이를 위해 필요한 자료 구조들을 더 정의할 것이다. 추가적으로, 현재 실행되고 있는 프로그램의 executable 파일을 수정하지 못하게 하도록 file\_deny\_write, file\_allow\_write 등을 활용해 쓰기를 제한한다. 모든 system call들을 filesys에 있는 함수들을 활용하는데, 이 함수들은 여러 쓰레드에서 동시에 calling하면 위험하기 때문에 synchronization을 적용하기도 할 것이다.

1. **개발 범위 및 내용**
   1. **개발 범위**

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

1. File Descriptor

System call에서 파일에 접근할 때 file descriptor을 쓰게 된다. 따라서 어떤 파일이 어떤 file descriptor이랑 연결이 되어있는지 저장을 해 둬야 한다. File descriptor을 구현하게 된다면 복잡한 포인터와 관련 없이 간단한 integer만으로도 원하는 파일에 접근을 할 수 있게 된다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

Create(name, initial\_size) : 새로운 파일을 initial\_size 크기로 만들게 된다. 이 syscall을 구현하면 syscall을 통해 새로운 파일을 생성할 수 있을 것이다.

Remove(name) : name에 해당하는 파일을 지운다. 이를 구현하면 syscall을 통해 존재하는 파일을 삭제할 수 있다.

Open(name) : name에 해당하는 파일을 연다. 이 syscall을 통해 파일을 열면, file descriptor을 할당받게 되는데, 나중에 이 file descriptor을 이용해서 파일과 관련된 여러 동작을 수행할 수 있다.

Close(file\_descriptor) : 더 이상 쓰지 않을 파일의 file descriptor에 해당하는 파일을 닫는다. 이를 구현하면 파일을 닫아서 시스템 리소스를 반환받을 수 있다.

Filesize(file\_descriptor) : file descriptor에 해당하는 파일의 크기를 반환받는다. 이를 구현하면 파일의 크기를 알 수 있다.

Seek(file\_descriptor, position) : file descriptor에 해당하는 파일의 위치를 position으로 지정한다. 이를 구현하면 파일의 임의의 부분으로 이동해서 동작을 할 수 있다. (read/write)

Tell(file\_descriptor) : 파일의 다음에 read/write할 위치를 알려준다. 이를 구현하면 어디까지 read/write를 했는지 알 수 있고, 다음에 read/write할 위치를 알 수 있다.

Read(fd, buffer, size) : 기존에 구현했던 stdout에 대한 read에 더해서 임의의 file descriptor에서 read를 진행할 수 있게 된다. 파일에 무슨 내용이 있는지 확인 할 수 있을 것이다.

Write(fd, buffer, size) : 기존에 구현했던 stdin에 대한 write에 더해서 임의의 file descriptor에 대해서 write를 진행할 수 있게 된다. 따라서 파일에 새로운 내용을 쓸 수 있게 된다.

3. Synchronization in Filesystem

Filesystem 관련된 코드를 (filesys/ 에 있는 코드) 실행할 때 여러 쓰레드에서 함수들을 동시에 calling하면 안전하지 않다. (pintos 매뉴얼에 따르면) 따라서 각 syscall을 구현할 때 filesys/ 에 있는 함수들을 쓸 때 lock을 걸어서 다른 쓰레드에서 부르지 못하게 한다. 또한 load 함수에서도 filesys에 있는 함수를 쓰기 때문에 lock으로 감싸줘야 한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

File descriptor을 저장하기 위해서 단순한 array를 쓸 것이다. File descriptor의 범위를 pintos 프로젝트에서는 128개로 한정하여, 정상적인 file descriptor은 0~127 중 하나의 값을 가지게 된다. 이 file descriptor이 array의 index로 들어가게 되면 빠르게 file 구조체 pointer을 찾을 수 있게 된다. 물론 128개의 파일 descriptor을 다 쓰지 않는다면 메모리 낭비가 있을 수 있지만, index로 접근하는 것이 속도 측면에서 이점이 있다고 생각하여 array로 구현하기로 했다.

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(filename, initial\_size): initial\_size 크기의 파일을 filename을 이름으로 해서 만들게 된다.

Remove(filename): filename에 해당하는 파일을 삭제한다.

Open(filename): filename에 해당하는 파일을 열고, 특정한 file descriptor을 할당한다. 이 descriptor들은 프로세스마다 고유하다.

Close(fd): file descriptor에 해당하는 파일을 닫게 된다.

Filesize(fd): file descriptor에 해당하는 파일의 크기를 되돌려준다.

Seek(fd, position): file descriptor에 해당하는 파일이 다음으로 읽거나 쓸 바이트를 position으로 위치를 변경한다.

Tell(fd): file descriptor에 해당하는 파일이 다음으로 읽을 바이트의 위치를 되돌려준다.

Read(fd, buffer, size): fd에 해당하는 파일의 내용을 size만큼 읽고, buffer에 넣어준다. 실제로 읽은 바이트를 리턴한다.

Write(fd, buffer, size): buffer에 들어있는 내용을 fd에 해당하는 파일에 size만큼 쓴다. 실제로 쓰여진 바이트 수를 리턴한다.

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

모든 쓰레드가 공통으로 사용하는 lock을 하나 만들고, filesys/ 에 있는 함수들을 부르기 전에 lock을 걸고, 부르고 나서 lock을 풀면 파일시스템 관련 동기화 문제는 해결할 수 있다. pintos에서는 lock\_acquire 함수로 lock을 걸고, lock\_release 함수로 lock을 풀게 된다. Lock은 lock을 건 쓰레드만 풀 수 있다는 점만 빼면 처음의 값이 1인 semaphore이랑 동일하게 동작 하기 때문에, lock 대신에 semaphore을 사용해도 된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

2023.10.25~2023.10.25: file descriptor table 구현

2023.10.26~2023.10.26: system call 구현

2023.10.27~2023.10.28: file system과 관련된 동기화를 구현

* 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

먼저, file descriptor을 저장할 array를 struct thread 안에 fd\_table이란 이름으로 넣었다. 최대 128개의 file descriptor만 저장하면 되기 때문에 크기는 128로 잡는다. 그리고 fd\_table init\_thread에서 초기화를 해준다. 나중에 filesys에 있는 함수들을 부를 때 동기화 문제를 해결하기 위해 thread.c에 lock을 하나 만들고, extern으로 선언해서 다른 파일에서도 쓸 수 있게 한다.

그 다음 system call을 구현하기 위해 syscall.c에서 syscall number에 해당하는 함수들을 만든다. 이 때 filesys/ 에 있는 함수들을 사용하기 전에 만들어둔 lock을 acquire하고, 사용이 끝나면 release를 해준다. Create를 구현하기 위해서 filesys\_create, remove를 구현하기 위해서 filesys\_remove를 쓴다. Open을 구현하기 위해서 filesy\_open을 쓰고, 오류가 없었다면 fd\_table에서 빈 공간을 찾아서 가장 낮은 file descriptor을 지정해준다. Close를 구현하기 위해 file\_close 함수를 쓴다. Filesize를 구현하기 위해 file\_length를 쓰고, seek를 구현하기 위해 file\_seek 함수를 쓴다. Tell을 구현하기 위해 file\_tell 함수를 쓴다. Read랑 write는 이미 구현되어 있는 것에 더불어서 file\_read, file\_write 함수들을 쓰게 된다. 이 때 file descriptor을 써야하는 모든 syscall들은 NULL 확인을 먼저 한 다음에 접근하고, 만약에 NULL이라면 오류를 리턴해야 한다.

Process.c에 있는 load 함수에서도 filesys에 있는 함수들을 쓰기 때문에 lock을 걸고 풀어줘야 한다. 또한 현재 실행이 되고 있는 프로세스에 해당하는 executable에 대한 write를 금지시키기 위해 file\_deny\_write 함수를 써서 write를 막는다. 하지만 현재 load 함수에서 file\_close를 하는데, 이를 하면 file\_allow\_write가 자동으로 된다. 이를 막기 위해 file\_close를 하지 않는다. 현재 executable에 대한 file을 따로 쓰레드에 저장해두고, 마지막에 exit을 부를 때 file\_close를 하게 바꾼다.

Exit을 할 때 모든 열린 파일을 닫아야하기 때문에 exit에 fd\_table에 열려 있는 파일들을 확인하고, 닫는 부분을 추가해야한다.

1. **연구 결과**
   1. **Flow Chart**

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

1. File descriptor

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

file descriptor은 struct thread 안에 있는 fd\_table에 저장한다. 위에 있는 flow chart들은 이 file descriptor들이 allocate, deallocate 되는 핵심 부분을 보여준다. 나머지 syscall들이 file descriptor을 어떻게 사용하는지는 아래 syscall에 대한 flowchart에 작성하였다.

2. System calls

각 system call들은 각각에 해당하는 함수를 부르게 된다. 그리고 그 안에서 어떤 함수들을 부르는지 flow chart를 그렸다.

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

3. Synchronization in filesystem

기본적으로 동기화를 적용하기 위해서 filesys/ 디렉토리 안에 있는 파일의 함수들은 lock을 걸고, 호출하고, 다시 lock을 푸는 형식으로 부르게 된다.

스크린샷, 폰트, 라인, 텍스트이(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, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명**

File descriptor을 저장하기 위해서 fd\_table이라는 배열을 struct thread 안에 넣었다. 배열의 크기는 128로 잡았다. pintos에서 파일 관련된 것은 struct file을 쓰기 때문에 이 포인터를 저장해 주었다. 또한 아래에 exec\_file을 따로 저장해 두었다. 이는 실행을 한 파일에 대한 포인터를 저장한 부분이다. (file\_deny\_write를 나중에 하기 위해)

2. System calls

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명

파일과 관련된 system call들을 각각 함수들을 불러서 각 함수에서 구현하게 되었다. 각 시스템 콜에 대한 내용은 아래에 서술하였다.

Create system call은 sys\_create에서 구현하였고, 먼저 스택으로부터 파일 이름을 저장하는 char\*를 얻고, initial\_size에 해당하는 unsigned를 얻는다. 그리고 lock을 acquire하고, filesys\_create 함수를 통해서 새로운 파일을 만든다. 리턴 값을 f->eax에 넣고, lock을 다시 release 해준다.

Remove system call은 sys\_remove에서 구현하였다. 스택으로부터 파일 이름을 가리키는 char\*를 얻어온다. 그 후 lock을 acquire하고, filesys\_remove 함수를 부른다. 이 함수의 리턴값을 eax에 넣고, lock을 release 해준다.

Open system call은 sys\_open에서 구현하였다. 스택으로부터 파일 이름을 가리키는 char\*를 file 변수에 저장하였다. 그리고 filesys\_open을 통해 struct file\*를 얻는다. NULL checking을 하고, 만약에 유효하다면 fd\_table에서 가장 낮은 file descriptor을 찾아서 열린 파일 포인터를 그 자리에 저장한다. 그리고 얻은 fd를 반환한다.

Close system call은 sys\_close에서 구현하였다. 스택으로부터 fd를 얻고, fd의 값을 확인한다. 그 다음 fd\_table에서 fd에 해당하는 struct file\*를 얻는다. File\_lock을 걸고 file\_close로 파일을 닫아준다. File\_lock은 다시 풀어주고, fd\_table도 초기화한다.

Filesize system call은 sys\_filesize에서 구현하였다. 스택에서 fd를 얻고, fd의 값이 범위 내에 있는지 확인한다. Fd\_table에서 fd에 해당하는 struct file\*를 얻어온다. Lock을 걸고 file\_length 함수를 통해 fd에 해당하는 파일의 크기를 얻는다. 그리고 이것을 eax에 저장해준다. 마지막에 lock을 release한다.

Seek system call은 sys\_seek에서 구현하였다. 스택에서 fd와 position을 pop한다. Fd의 범위가 적절한지 확인하고 fd\_table에서 해당하는 fd의 file 구조체 포인터를 얻어온다. Lock을 걸고, file\_seek 함수를 통해 struct file\*이 가리키는 파일의 position을 변경해준다. 마지막에 lock을 다시 풀어준다.

Tell system call은 sys\_tell에서 구현하였다. 스택에서 fd를 pop한다. Fd의 범위를 확인하고, fd\_table에서 struct file\*를 얻어온다. Lock을 걸고 file\_tell 함수를 통해 파일의 현재 위치를 얻는다. 이를 eax에 저장하고 다시 lock을 풀어준다.

Write system call은 이미 sys\_write에 일부가 구현이 되어 있었다. Fd가 1인 경우는 stdout에 출력되게 구현이 되어 있었는데, 나머지 경우에 대한 구현을 추가하였다. Fd\_table에서 fd에 해당하는 struct file\*를 얻는다. 그리고 lock을 걸고 file\_write 함수를 통해 buffer의 내용을 파일에 write한다. Eax에 write한 바이트 수를 넣어주고, lock을 release해준다.

Read system call은 이미 sys\_read에 구현이 일부 되어 있다. Fd가 0인 경우, stdin에서 입력을 받아 buffer에 저장하게 되어있다. 나머지 경우에는 fd\_table에서 struct file\*를 얻어온다. Lock을 걸고, file\_read를 통해 파일에 있는 내용을 buffer에 넣어준다. Eax에 읽은 바이트 개수를 넣어주고, lock을 release해준다.

3. Synchronization in Filesystem

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

Thread.h에 file\_lock이라는 lock도 추가해 주었다. Thread.c에 선언을 하였고, thread\_init에서 초기화를 진행한다. 이 lock을 써서 filesys/에 있는 함수들을 부르기 전에 acquire하고, 호출이 끝난 뒤에 다시 release해주었다.

현재 실행되고 있는 파일에 대한 write를 허용하지 않기 위해 load 함수에서 file\_deny\_write 함수를 써주었다. 그리고 thread struct에 열어 놓은 파일을 기록 해 두었다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Load 함수에 있는 file\_close를 삭제하고, 나중에 process\_wait를 할 때 열려있는 다른 file descriptor을 닫아줄 때 같이 close를 해주었다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

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

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명