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

담당 교수 / 분반 : 김영재 교수님 / 1분반

이름 / 학번 : 김동건 / 20211507

개발 기간 : 2024.10.28 ~ 2024.11.01

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

Pintos에서 file system과 관련한 System call들을 구현한다. create, remove, open, close, filesize, read, write, seek, tell이다. 이 과정에서 lock과 semaphore를 이용하여 synchronization과 관련한 문제가 없도록 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  
File Descriptor를 구현하면 파일을 열고 닫을 수 있게 된다. 따라서, 프로세스마다 원하는 파일에 입력과 출력을 하게 될 수 있다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

Project 1에서는 read와 write에서 각각 STDIN, STDOUT만 고려하였었다. Project 2에서는 STDIN, STDOUT 뿐만 아니라 다른 파일에 입출력 하는 것도 구현하여야 한다. 이 외에도 create, remove, open, close, filesize, seek, tell를 구현하여 OS가 파일 시스템을 구현할 수 있게 된다.

3. Synchronization in Filesystem

Pintos에서 여러 프로세스들이 같은 파일에 접근했을 때, 원하지 않는 값을 읽거나 쓸 수 있기 때문에 읽기와 쓰기를 할 때에는 synchronization 문제가 일어나지 않게 lock 활용하여 구현한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

정적 배열을 이용하여 File Descriptor를 구현하였다. 128개까지만 가질 수 있다고 하였지만, 여유 있게 135칸을 잡아 주었다. 정적 배열을 사용한 이유는, 이번 프로젝트에서 최대 파일 개수가 정해져있기 때문에 굳이 동적을 사용할 필요가 없었기 때문이다. 만약 동적으로 이를 관리한다면 메모리 누수를 신경 써야 하므로 사용하지 않았다. 이는 각 thread마다 관리될 수 있게 thread 구조체 내에 추가하였다.

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: file을 생성하는 syscall이다. 생성 성공 실패 유무를 bool 값으로 return한다.

remove: file을 삭제하는 syscall이다. 삭제 성공 실패 유무를 bool 값으로 return한다.

open: file을 여는 syscall이다. open한 file descriptor 값 혹은 open에 실패한 경우 -1을 return한다.

close: file을 닫는 syscall이다. close할 파일이 없는 경우라면 exit이 된다.

filesize: file의 size를 반환하는 syscall이다. 열려있지 않은 파일이라면 exit이 된다.

read: 주어진 file descriptor에 해당하는 파일에서 size만큼 읽는다. 읽을 수 없는 경우라면 exit이 된다.

write: 주어진 file descriptior에 해당하는 파일에 size만큼 쓴다. 쓸 수 없는 경우라면 exit이 된다.

seek: pos만큼 현재 file의 포인터를 변경한다.

tell: file descriptior가 가르키는 파일의 포인터를 return한다.

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

같은 파일에 쓰기가 진행될 때 이를 방지하기 위해 write에서 lock을 진행해 준다. 처음에는 write에만 lock을 걸어도 모든 케이스가 통과하였으나, write뿐만 아니라 파일과 관련된 syscall이 진행될 때에는 파일의 변경이 있으면 안 될 거라 판단하여, file과 관련된 syscall 전체에 lock을 사용하였다.  
 다음으로, 스레드에 load\_lock이라는 세마포어도 추가하였는데, 이는 부모 자식 스레드 간의 기다림을 관리할 수 있게 하기 위함이다. 자식이 로드되기 전에 부모가 죽는 경우를 방지하기 위함이다. 만약 그런 경우가 있다면, 자식 스레드가 파일에 접근하려 할 때, 부모가 먼저 죽어 load가 되지 않으면 비정상적인 접근이 일어나는 등의 문제가 생길 수 있기 때문에 자식의 load를 기다려주는 세마포어를 하나 사용하였다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성
* 2024.10.26: 핀토스 매뉴얼 공부
* 2024.10.27 ~ 2024.10.28: file관련 syscall들 구현 및 file\_deny\_write를 통한 rox 케이스 해결, synchronization을 통해 syn - read, write 케이스 해결
* 2024.10.28: wsl을 이용하여 local에 pintos 개발 환경 구축
* 2024.10.29 ~ 2024.10.31: multi-oom 케이스 해결을 위한 디버깅 및 디버깅 과정 기록
* 2024.11.01: 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

우선, thread내에 load를 기다려주는 세마포어, 파일 디스크립터 배열, wait\_status가 추가되어야 하므로 thread.h와 thread.c가 수정되어야 한다. 또, file관련 syscall들을 구현해야 하므로 syscall.c가 수정되어야 한다. 이때, 파일에 대한 synchronization 문제를 다루기 위한 lock 구조체를 syscall.c에 선언하여 파일 관련 함수들에서 사용한다. 이전에 구현하지 않았던, 파일 관련 syscall들인 create, remove, open, close, filesize, seek, tell 함수를 syscall.c에 구현한다.

1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 3가지 항목에 대하여 Flow Chart 작성
  + 1. File Descriptor  
       **텍스트, 라인, 도표, 스크린샷이(가) 표시된 사진

       자동 생성된 설명**
    2. System Calls  
       **도표, 평면도, 라인, 평행이(가) 표시된 사진

       자동 생성된 설명**
    3. Synchronization in Filesystem  
       Synchronization을 위해 첫 번째로는 syscall 들에서 lock을 사용하였다. 이는 2번 문항의 플로우차트에서 확인할 수 있다. 그 외에 load\_lock이라는 세마포어와 wait\_status를 통해 부모-자식 스레드 간의 synchronization 문제를 해결하였다. 해당 플로우 차트는 아래와 같다.  
       **텍스트, 도표, 기술 도면, 평행이(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

가장 먼저 파일과 관련한 syscall을 구현하기 위해 thread에 fd배열을 추가하였다. 

이후에 syscall.c에 파일 관련 syscall들을 구현하였다. syscall\_handler에서 address를 검사하고 syscall 호출하는 것은 프로젝트1과 같은 방식으로 진행하였다. 텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

exit syscall에서 메모리 누수가 발생하지 않게, 파일이 존재하면 close해주는 코드를 추가하였다.

아래 syscall들은 모두 file과 관련된 syscall이다. 공통된 부분인 lock\_acquire를 먼저 설명하겠다. 처음에는 lock을 없이 구현하였더니, syn관련 테스트 케이스들을 통과하지 못하여서 뒤늦게 추가한 부분이다. 우선, filesys\_lock이라는 lock 구조체를 syscall.c에 추가하였다. 이는 파일에 관한 syscall이 진행될 때, 다른 프로세스들이 접근을 하지 못하게 막아주는 역할을 할 때 쓰이는 lock이다. 가장 먼저 syscall\_init 함수에서 filesys\_lock을 초기화 해준다. 텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

이후에, 각 syscall 시작 단에서 lock\_acquire(&filesys\_lock)을 통해 lock을 설정하고, syscall이 종료되기 전에 lock\_release(&filesys\_lock)을 통해 lock을 해제해 준다. 이렇게 synchronization 문제를 해결해 주었다. 이제 각 파일 syscall들을 하나씩 살펴보자.

**텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

read 함수이다. 기존에 stdin인 경우에만 구현이 되어 있었고, 추가로 fd != 0일 때도 구현해 주었다. 만약 fd값이 범위를 벗어나거나, 현재 스레드에 해당 파일이 없는 경우 예외 처리를 해주었고, 그렇지 않은 경우에는 file\_read를 통해 read작업을 수행한다.

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

write도 read와 같은 구조를 갖게 구현하였다. 함수가 file\_read 대신 file\_write인 점만 다르다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

create와 remove이다. 각각 인자가 NULL인경우 예외처리를 하였고, 그 외의 경우에는 함수를 실행하게 작성하였다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명open 함수이다. 파일이 없는 경우 -1이 리턴되게 하였고, 그렇지 않은 경우엔 비어있는 첫 fd를 배정하여 주었다. 이때, strcmp(file, thread\_current()->name)가 0인 경우 file\_deny\_write(fp)를 해주었다. 이 함수는 과제 명세서에 나와 있던 함수로, write를 금지시키는 함수이다. 이것을 해준 이유는 rox 관련 테스트 케이스들을 해결하기 위함이었다. 왜냐하면 rox 테스트 들은 실행중인 프로세스들이 수정되는 것을 확인하는지 보는 케이스들이기 때문이다. 따라서 만약 현재 프로세스가 open되는 file과 같은 경우 수정되는 것을 막는 것이다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

filesize, seek, tell, close는 어렵지 않게 구현하였다. 주어진 값이 NULL인 경우 예외로 처리하였다. 다만, close의 경우 바로 close를 하니 close twice 테스트 케이스가 통과하지 않았다. 왜냐하면 파일을 닫기만 하고 배정된 fd 배열에서는 삭제하지 않았기 때문이다. 따라서 close하는 fd는 스레드의 fd 배열에 NULL로 변경해 두어 두 번 닫히는 일이 없게 방지하였다.

마지막으로 multi-oom 케이스가 통과하지 않았다. 이유는 모르지만 begin()이 계속되는 현상이 나타났다. 케이스 설명을 읽어보니 fork를 계속 반복한다고 한다. 우선, 포크를 계속 진행하면서 메모리 누수가 나면 안 되기 때문에 기존에 process.c load부분에서 파싱을 할 때 malloc 한 메모리를 잘 free 해 주었는지, 파일은 잘 닫았는지 확인해 주었다. 이 과정에서 file을 닫지 않았던 것을 잘 닫아주었다. 그래도 문제는 해결되지 않았다.

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명



multi-oom.c에서 n이 처음에만 0이고 그 후에는 0이 아니어야 하는데, 디버깅을 해보니 계속 n이 0이 되어 첫 프로세스일 때만 begin이 출력되어야 하는데, 계속 begin()이 출력되는 것이었다.

텍스트, 폰트, 스크린샷, 친필이(가) 표시된 사진

자동 생성된 설명

multi-oom.c에 디버깅 코드를 추가하였는데, 놀랍게도



첫 인자에 빈 문자열이 있다. 프로젝트 1에서 파싱과 관련된 케이스들은 모두 통과하였기 때문에 문제가 없다고 생각했는데, 아니었던 것 같다. 알고 보니 load 함수에서 파싱을 하는 과정에서 맨 끝 문자가 공백인 경우를 고려하지 않았었고, 기존 project 1 테스트 케이스들에서는 그러한 케이스가 없어 문제가 되지 않았던 것이다. 그러나, 이번 multi-oom 케이스에서는 맨 끝이 공백인 경우가 있어 문제가 되었다. 따라서 project1에서 했던 파싱 코드를 수정하여 맨 끝 공백도 처리를 잘 하게 수정하였다. 그럼에도 multi-oom 관련해서 wait 관련 오류가 발생하였다.

부모가 자식을 생성하였는데, 자식이 온전히 load 되기 전에 부모가 죽어버리는 경우에 자식이 값을 온전히 load하지 못해서 문제가 생긴다. 이를 해결하기 위해 load\_lock이라는 세마포어를 스레드에 도입하여 create 직후에 sema를 down한다. thread\_create에서 start\_process를 호출하고, 여기서 load가 다 된 경우에 sema를 up한다. 이때, 부모의 세마포어가 무엇인지 알 수 없다. 따라서 parent thread 주소도 thread 구조체 안에 추가하여야 한다. 즉, 정리하자면 부모-자식 간의 load에 문제가 없게 하는 세마포어를 하나 생성하였고, 자식이 부모의 세마포어에 접근할 수 있으려면 부모 스레드의 주소를 갖고 있어야 하니 부모 스레드의 주소 역시 구조체에 추가하였다.





thread.c에서 init이 될 때 해당 변수들도 아래와 같이 init해주어야 한다.

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

list\_init은 project 1부터 있던 기존 코드이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위 사진처럼 load\_lock과 관련한 세마포어를 통해 자식이 load가 다 되기 전에 부모가 진행되지 않도록 하였다. 이제, start\_process에서 load함수가 호출되고, load가 다 되었으니본인 부모의 load\_lock 세마를 up하여 준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

여기까지 진행했음에도 multi-oom 테스트는 통과되지 않았다. load failed 와 같은 오류가 종종 발생한다. 자식이 load에 실패되었을 때 exit만 하는 것이 아니라, 부모의 child list에서 삭제시켜야 한다. 해당 스레드들은 부모의 wait을 기다리고 있을 것이다. 이를 확인하기 위해, wait\_status를 구조체에 추가하였다. 프로젝트 1에서 exit 여부를 저장하는 exit\_status와는 다른 변수임에 주의하자.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

만약 load를 실패한다면 wait\_status를 켜주어 표시를 해주자. 그 후, 다시 부모 스레드에서 execute를 끝내기 전에 그러한 프로세스들을 찾아 process\_wait을 호출해 주자.

텍스트, 소프트웨어, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

이제 모든 테스트 케이스들이 통과된다.

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

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명80개의 test가 모두 통과되는 것을 확인할 수 있다.