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

담당 교수 / 분반 : 박성용 교수님 / 1반

이름 / 학번 : 남현준 20181625

개발 기간 : 10/9 ~ 10/29

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

Pintos project #1 에서 User program에서의 system call, system call handler, user stack의 run-time 생성을 구현했지만 아직 파일과 관련된 system call의 경우, system call handler에서의 구현이 되어있지 않다.

따라서, pintos project #2에서의 목표는 project #1에서 구현하지 않은 나머지 system call에 대해 관련 함수와 system call handler에서의 작동을 구현하게 될 것이고 이에 따라 파일의 read & write에서 발생할 수 있는 synchronization 등의 문제도 같이 해결하여 OS로서의 기본적인 기능들이 모두 구현될 수 있도록 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

File Descriptor는 process가 file을 open system call을 통해 열게 될 경우, 해당 file을 식별하고 해당 정보를 저장한 다음 사용해야 할 때 어떤 file을 사용하는지 알려줄 수 있게 된다. Pintos의 경우, 현재 file descriptor의 구현이 되어있지 않고 이에 따라 process가 실행되며 어떤 file을 사용하게 될 것인지 식별할 수 있는 방법이 없다고 할 수 있다.

File descriptor를 구현하게 된다면 file을 open system call을 통해 open했을 때, 해당 정보를 file descriptor에 저장하고 후에 open, read 혹은 write system call을 user가 invoke했을 때 정상적으로 필요한 파일의 내용을 읽거나 수정하는 것이 가능해질 것이다.

File descriptor의 경우, Pintos manual에 최대 128개까지의 file descriptor를 저장할 수 있다고 명시되어 있고 이에 따라 file descriptor를 저장할 struct \*file type의 배열을 고정된 크기 128개로 선언할 것이다. 운영체제 강의자료의 내용에 따라 file descriptor의 0, 1, 2번째 index는 각각 STDIN, STDOUT, STDERR에 할당될 것이므로 구현할 file descriptor의 3번째 index부터 실행하는 파일의 정보가 저장될 수 있도록 할 것이다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

Project #2에서는 project #1에서 구현하지 않은 나머지 system call에 대해서 구현하게 될 것이다.

해당되는 system call로는 create, remove, open, close, filesize, seek, tell과 같은 system call이 있고 user가 이러한 요청을 했을 때 system call handler를 통해 정상적으로 system call을 handle할 수 있게 구현되어야 할 것이다.

3. Synchronization in Filesystem

File의 경우 synchronization을 진행하지 않을 시 문제가 발생할 수 있다. Process가 실행되며 생성되는 thread의 경우, process의 자원을 공유하기 때문에 여러 thread에서 특정 file을 읽는 것과 동시에 특정 file이 내용을 수정할 수 있게 된다. 이러한 문제는 매우 치명적인데 특정 file을 읽는 thread와 해당 file을 수정하는 thread가 동시에 실행된다면 실제 읽어야 하는 내용은 수정되기 이전의 내용인데 수정한 다음의 내용을 읽어서 사용하게 되는 불상사가 발생할 수도 있게 된다는 것이다.

따라서, 해당 문제는 synchronization을 통해 관리를 할 필요성이 있고 위에 명시된 reader-writer synchronization problem을 해결하기 위해 semaphore, lock 등과 같은 synchronization을 위한 구조체를 활용하여 read와 write가 한 파일에서 동시에 일어날 수 없도록 할 것이다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

Thread의 initialization 진행 시, 각 thread에 struct \*file fd[128]을 통해 동일하고 고정된 크기의 file descriptor를 저장할 fd 배열을 선언했고, fd를 모두 NULL로 초기화했다.

배열을 사용한 이유는 pintos manual에서 이미 최대 128개의 file이라고 명시되어 있고, 운영체제 강의자료에서 설명한 file descriptor의 작동과 비슷하다고 생각됐기 때문이다. 강의자료에서 file descriptor는 file이 close될 때, 해당 file descriptor를 NULL로 설정하고, 새로운 file을 open하는 경우 file descriptor의 번호 중 가장 작으면서 NULL인 fd의 번호에 file의 정보를 연결하기 때문이다.

또한, 배열을 사용해서 구현할 file descriptor에서 insert와 delete에 걸리는 시간이 O(N)이지만 최대 크기가 128로 insert/delete에 강점이 있는 자료구조로 구현하는 것과 소모 시간에 있어 유의미한 차이를 보일 것 같지 않기 때문에 구현이 직관적이면서도 간단한 배열을 활용해서 file descriptor를 구현하였다.

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

File의 creation에 사용되는 system call. 사용되는 parameter가 invalid한 memory address를 접근하거나 file name의 string이 NULL일 경우 exit(-1)등과 같은 작동을 통해 handle 될 것이다.

Remove

File의 remove에 사용되는 system call. 사용되는 parameter가 invalid한 memory address를 접근하거나 file name의 string이 NULL일 경우 exit(-1)등과 같은 작동을 통해 handle 될 것이다.

Open

파일 이름을 받아 해당 file을 open 하도록 요청하는 system call. 이름과 일치하는 file이 없다면 exit(-1) 등과 같은 방법으로 error를 handle. Open된 파일에 대해서는 수정이 불가능하도록 file\_deny\_write() 등과 같은 함수를 통해 수정되지 못하도록 설정하도록 한다,

Close

특정 file descriptor에 open된 파일이 있다면 해당 file을 close하도록 요청하는 system call. File descriptor가 NULL이라면 exit(-1)과 같은 방식으로 error를 handle.

Read

Read system call을 통해 특정 file descriptor에 열려 있는 file의 내용을 read할 수 있게 될 것이다. 이 과정에서, reader-writer problem이 일어나지 못하도록 synchronization과 관련된 작동이 들어가야 할 것이다.

Write

Write system call을 통해 STDOUT을 제외하고도 file을 write가 가능해지도록 구현한다. 이 때, reader-writer problem이 일어나지 않도록 read 중인 file에는 write가 진행되지 못하도록 synchronization과 관련된 작동을 구현하게 될 것이다.

Filesize

File descriptor의 크기를 계산하고 return하게 되는 system call이다.

Seek

File에서 시작위치로부터 new\_pos byte만큼 떨어진 위치를 current position으로 set 하는 system call이다.

Tell

File의 시작위치로부터 몇 byte만큼 떨어져있는지 해당 결과를 return하는 system call이다.

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

Lock과 semaphore의 경우 사용하기에 앞서 우선적으로 선언된 다음 initialization을 진행하야 pintos 내 프로그램의 실행에서 필요한 상황에 사용할 수 있게 된다.

Lock의 경우 해당 구조체가 src/threads의 synch.h에 정의되어 있고 이를 바로 system call handler가 정의되어 있는 src/syscall.c에서 사용하려고 할 경우, compile 상황에서 에러가 발생하게 된다. 따라서, system call handler를 initialize할 때 lock 또한 해당 파일에서 lock을 initialize하는 함수인 lock\_init()을 통해 initialization을 진행하게 될 것이다.

Lock을 사용해야 하는 상황에서는, 해당 코드가 실행되기에 앞서 lock\_acquire()함수를 통해 lock을 생성하여 다른 process에서 해당 file에 접근할 수 없도록 한 다음 특정 함수에서 작성한 파일과 관련된 작동을 하도록 한다. 파일의 사용이 끝났다면 lock\_release()를 통해 lock을 해제하여 다른 process에서 사용될 수 있도록 한다.

Semaphore의 경우, src/threads의 thread.h 파일에 semaphore와 관련된 내용을 작성한 다음, thread.c 파일의 thread\_init()과 init\_thread() 함수에서 semaphore의 initialization과 관련된 작동을 하게 된다. 이후, 작성한 semaphore를 사용해야 하는 경우, sema\_up() 함수와 sema\_down()을 이용하여 semaphore의 값을 변경하게 된다. 이 때, semaphore의 값이 0이고 sema\_down()을 사용하는 경우, semaphore를 0보다 낮게 설정할 수 없으므로 어느 부분에서 sema\_up()을 통해 semaphore의 값에 변화가 생길 때까지 대기하게 된다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

File Descriptor : 10/9 ~ 10/11

System call : 10/11 ~ 10/29

Synchronization : 10/13 ~ 10/29

보고서 작성 : 10/26 ~ 10/29

* 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

1. File descriptor

File descriptor의 구현을 위해 src/threads의 thread.h 파일에 struct \*file fd[128]을 선언했다. 해당 file type의 fd 배열이 file descriptor의 정보를 저장하게 될 것이다.

이후, 같은 폴더 내의 thread.c의 init\_thread() 함수에서 각 thread에서 STDIN, STDOUT, STDERR에 사용할 0, 1, 2 번째 index를 제외하고 나머지 file descriptor들은 아직 사용하지 않기 때문에 NULL로 initialization을 진행하도록 한다.

1. System calls

Syscall.c 파일에서 open, close, read, write, filesize, seek, tell 등의 system call에 대응할 수 있게 syscall.c에 관련 함수를 추가적으로 작성한다. 이 때, file과 관련된 함수의 library인 filesys/file.h에 정의된 함수를 사용할 수 있도록 헤더 부분에 filesys/file.h 파일을 include하여 사용할 수 있도록 한다.

헤더 파일과 사용할 함수의 정의를 마쳤다면 system call handler의 switch문에서 해당 system call의 번호에 맞는 요청이 오게 될 때, 요청한 타입의 system call에 반응할 수 있도록 case별로 작성한 함수를 요청하도록 하거나 intr\_frame \*f의 eax에 return 값이 저장될 수 있도록 한다.

1. Synchronization in filesystem

Synchronization을 위해서는 lock이나 semaphore등을 선언하여 사용할 필요가 있다. 따라서, 해당 내용이 정의되어 있는 synch.h 파일을 헤더로 include할 필요가 있다. 해당 구조체를 사용할 syscall 부분과 process 부분에서 사용할 수 있도록 헤당 헤더 파일을 include한다.

헤더 파일의 include가 진행되었다면 pintos manual의 요구사항에 맞춰 작동될 수 있도록 특정 함수들의 수정을 진행해야 한다.

우선 process가 시작되기 전, process의 load가 fail이 되어 종료가 되어야 할 경우, child thread들이 존재할 때 해당 thread의 exit status를 process\_wait() 함수를 통해 reaping이 진행될 수 있도록 해야 한다.

Pintos에서 open이 진행된 file에는 수정을 할 수 없도록 조치를 취해야 한다. 이를 위해, process의 load를 진행하며 load에 성공했다면 해당 파일에는 write를 진행할 수 없도록 file\_deny\_write() 함수를 호출하여 write를 disable시킬 수 있도록 한다.

Exit system call을 받게 되었을 때는 thread를 종료시키게 되는데 이 때, 해당 thread에 열려 있는 file descriptor들을 file\_close() 함수 등을 이용해 닫힐 수 있도록 한다.

Syscall 구현 부분에서는 lock을 통해 synchronization 문제를 해결할 수 있도록 한다. Lock을 사용할 수 있도록 src/threads/synch.h 헤더 파일을 include했다면 system call handler를 initialize하며 lock 또한 initialize를 진행한다.

이후에, file을 읽거나 사용해야 할 때, lock\_acquire()를 통해 lock을 생성하여 한 번에 하나의 접근만 이루어질 수 있도록 구현한다. 파일과 관련된 작동을 다 한다음에는 lock\_release()를 통해 다음 파일에 대한 접근이 진행될 수 있도록 한 다음 결과물을 return하도록 구현한다.

1. Exception.c

Src/userprog의 exception.c을 수정하여 요청한 page가 존재하지 않을 때 종료될 수 있도록 해야한다. 이를 위해 project #1에서 수정한 내용에 추가적으로 bool not\_present가 true일 때 exit(-1)을 통해 종료될 수 있도록 수정을 진행한다.

1. **연구 결과**
   1. **Flow Chart**

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

1. File Descriptor

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

1. System call

스크린샷, 텍스트, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명

1. Synchronization

스크린샷, 텍스트, 도표, 디자인이(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
* File descriptor

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Thread에서 file descriptor의 정보를 관리할 수 있도록 struct file \*fd[128]을 선언했다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Thread의 생성 시, initialization을 진행할 때, thread.h에서 선언한 struct \*file type의file descriptor 배열을 NULL로 초기화한다.

Project #1에서 사용한 semaphore 이외에 project #2에서의 구현을 위해 추가적으로 선언한 loadThread semaphore 또한 초기화를 같이 진행한다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

Exit system call에서 thread가 종료될 때, 열려 있던 STDIN, STDOUT, STDERR을 제외한 file descriptor들이 닫힐 수 있도록 close() 부분을 추가했다.

* System call
* Synchronization in filesystem

(System call과 synchronization에 대한 코드 부분과 설명은 같이 넣었습니다)

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

System call handler에서 pintos project #2에서 구현해야 하는 system call인 create, remove, open, close, filesize, read, write, seek, tell의 system call 번호가 주어졌을 때 handler를 통해 처리할 수 있도록 switch문에 해당 case들을 추가해주었다. 추가한 모든 system call에 대해서는 is\_user\_vaddr() 함수를 이용해 esp에 저장된 system call에서 사용할 parameter들이 valid한 주소에 있는지를 판단하고 아닐경우 exit(-1)을 통해 종료될 수 있도록 했다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Create system call에서 사용되는 함수의 코드 부분이다. Parameter로 주어지는 file의 이름이 Null string일 경우, exit(-1) 함수를 통해 종료될 수 있도록 하였다. 그 외의 경우에는 src/filesys의 filesys.h와 filesys.c 파일에 정의되어 있는 filesys\_create() 함수를 사용하여 file의 생성을 진행한다. 이후, 생성에 성공했다면 해당 함수에서는 true를, 실패했다면 false를 return한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Remove system call에서 사용되는 함수의 코드 부분이다. Parameter로 주어지는 file의 이름이 null string일 경우 exit(-1)를 통해 실행이 종료될 수 있도록 한다. 그 외의 경우에는, filesys\_remove()함수를 사용하여 parameter로 주어진 파일과 같은 이름을 가지는 file이 있는지 검사하고 삭제를 진행한다. 삭제에 성공하였다면 true를, 아니라면 false를 return한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Open system call에서 사용되는 함수이다. Parameter로 주어지는 string이 null일 경우 exit(-1)을 통해 실행이 종료될 수 있도록 한다.

Parameter로 주어진 string이 null string이 아니었다면 lock\_acquire()를 통해 lock을 생성한다. Lock을 생성하는 이유는 open system call을 처리할 때 다른 open system call이 호출되어 간섭될 수 없도록 하기 위함이다.

Filesys\_open()을 통해 해당 file을 open 해본다. Valid한 파일이고 실제로도 open이 되었다면, 이후에는 STDIN, STDOUT, STDERR에 사용될 0, 1, 2번째 index를 제외하고 file을 open한 다음 해당 정보가 저장될 수 있는 공간이 file descriptor 배열에 남아있는지 탐색을 진행하며 검사한다. 만약, file descriptor가 NULL인 index가 있었다면 해당 file descriptor 배열의 index에 open한 파일의 정보를 assign을 통해 link한다. 그 후, lock\_release()를 통해 lock을 해제한 다음 해당 file descriptor가 할당된 배열의 index를 return한다.

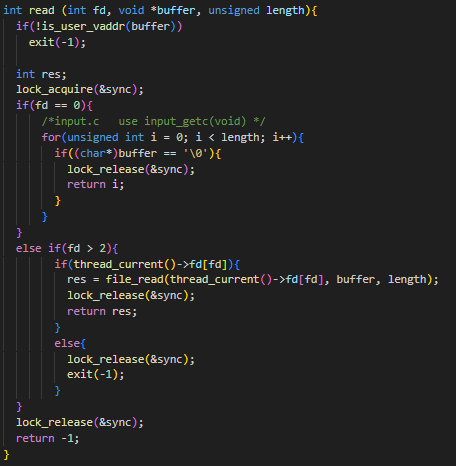
만약 file descriptor 배열에 비어 있는 공간이 없었거나 filesys\_open()을 통해 open한 파일이 제대로 된 실행파일이 아니었다면 lock\_relase()를 통해 lock을 풀고 -1을 return한다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

Filesize system call에 사용되는 함수이다. Parameter로 file descriptor의 index가 주어진다. 현재 실행중인 thread의 file descriptor 배열에서 해당 index의 file descriptor가 사용 중이 아니라면 exit(-1)을 통해 종료될 수 있도록 한다.

그 외의 경우에는 해당 file descriptor에 연결되어 있는 파일의 길이를 file\_length() 함수를 통해 계산하고 그 값을 결과물로 return한다.



Read system call에 사용되는 함수이다. Parameter로 fd, buffer, length가 주어진다.

시작하기에 앞서 lock\_acquire()을 통해 lock을 생성한다. 이는, 해당 파일과 다른 동시에 일어나는 작업을 막기 위해 생성하는 것이다.

Parameter로 주어진 fd의 번호에 따라 작동이 나뉘게 된다. STDIN이었을 경우 내용을 읽은 다음 lock\_release()를 통해 lock을 해제하고 그 때 STDIN에서 읽어들인 길이를 return한다.

Fd가 3 이상일 경우, 현재 thread의 file descriptor 배열에서 해당 번호를 가진 file descriptor가 유효한 상태인지 확인하고 file\_read()를 통해 파일을 읽는다. 그 후, lock\_release()를 통해 lock을 해제하고 file\_read()를 통해 얻은 값을 결과로 return한다. 유효하지 않았을 경우 lock을 해제하고 exit(-1)을 통해 종료될 수 있도록 한다.

그 외에 STDOUT이나 STDERR에 해당되는 fd였을 경우 -1을 return한다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Write system call에 사용되는 함수이다. 우선, buffer의 위치를 확인한다. Buffer가 invalid한 위치에 있을 경우 exit(-1)을 통해 종료될 수 있도록 한다.

Lock\_acquire()을 통해 lock을 생성한다. File에 write를 할 때 다른 작업이 이뤄질 수 없도록 설정하는 것이다.

Parameter로 받은 fd번호가 STDOUT에 해당되는 1이었다면 putbuf()를 통해 내용을 STDOUT에 write한 다음 lock\_release()를 통해 lock을 해제한다. 그 후, 그 때 출력한 buffer의 길이를 결과로 return한다.

fd번호가 3 이상이었다면 해당 번호의 file descriptor가 유효한지 확인한다. 유효할 경우 lock\_release()를 통해 lock을 해제하고 file\_write()함수를 통해 얻는 결과를 return한다. 유효하지 않을 경우 lock을 해제하고 exit(-1)을 통해 종료될 수 있도록 한다.

STDIN이나 STDERR에 해당되는 fd였을 경우 lock\_release()를 통해 lock을 해제하고 -1을 return한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Seek, tell, close system call에서 사용되는 함수들이다. 각각 parameter로 전달되는 fd번호의 file descriptor가 유효한지 확인한다. 유효하지 않을 경우 exit(-1)을 통해 종료될 수 있도록 한다.

유효할 경우 seek()에서는 file\_seek() 함수에 정의된 내용대로 작동될 수 있도록 한다.

Tell()에서는 file\_tell()함수에 정의된 내용대로 작동될 수 있도록 한 후 해당 결과를 return한다.

Close()의 경우 close\_twice test를 통과할 수 있도록 해당 file descriptor를 NULL로 설정한 후 다시 한번 file\_close()를 통해 해당 file descriptor를 닫을 수 있도록 함수를 구현했다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

해당 부분은 process를 load하는 과정에서 user stack을 생성하는 부분이다. 만약 load가 성공적으로 끝났다면 해당 프로그램은 정상적으로 process로서 실행됐음을 의미하므로 해당 file의 내용에 수정이 진행될 수 없도록 해야한다. 따라서, stack의 구현이 끝난 다음, file\_deny\_write() 함수를 호출 현재 실행중인 file에 대해 write()를 불가능하도록 설정했고 해당 정보를 기억하기 위해 thread에서 선언된 runningFile에 이 정보를 저장한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Process\_execute 함수에서도 수정을 진행했다.

Thread를 생성하기에 앞서 열려고 하는 file이 invalid한 파일이라면 해당 파일을 file\_close() 함수를 통해 닫은 다음 -1을 결과로 return하도록 했다.

Thread에 생성에 있어 sema\_down()을 통해 create된 thread의 정보가 sema\_up()이 호출되기 전까지 유지될 수 있도록 했다. 이 중, 특정 thread에 waitStatus가 활성화 되어 있을 경우, 종료가 될 때까지 기다릴 수 있도록 process\_wait()를 통해 wait한다.

waitStatus가 활성화되어 있지 않은 경우 해당 thread의 tid를 return한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Start\_process() 함수를 통해 process가 load되어 thread에서 시작될 load가 끝난 thread에 대해 sema\_up()을 사용한다. 만약 load가 제대로 이루어지지 않았다면 현재 실행 중인 thread의 waitStatus를 1로 set하여 앞서 설명했듯 회수될 수 있도록 한다.

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

텍스트, 스크린샷, 패턴, 패브릭이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 메뉴이(가) 표시된 사진

자동 생성된 설명

Sc-boundary-3

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Create-long

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Open-twice

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Read-bad-fd

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

출력 밀림으로 인한 ExExecution.. 출력 fail

Write-bad-fd

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Read-bad-fd와 동일한 이유로 fail

Bad-read2

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

동일한 이유로 fail