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

담당 교수 / 분반 : 김영재

이름 / 학번 : 임나현 / 20211582

개발 기간 : 2024.10.26-2024.10.31

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술
* Pintos의 base file system을 이용하여 prj1에서 구현한 system call 이외에도 create, open, close 등의 새로운 system call을 구현한다. File descriptor에서 적절한 위치를 찾아 파일을 처리해야 하며, 여러 thread 간의 동일한 파일에 대한 요청에도 서로 간섭 받지 않고 작동할 수 있도록 semaphore나 file\_deny\_write 함수를 적절히 사용하여 구현해야 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

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

1. File Descriptor

argument로 주어지는 file\_name을 stack에 공백 단위로 저장하고 관리하기 위해서는 먼저 parsing 과정이 필요하다. Argument을 공백을 기준으로 이차원배열에 동적 할당하여 저장한다. 그 후, stack에 순서대로 passing하여 저장한다. hex\_dump를 사용하면, stack에 제대로 저장이 되어 있는지 확인할 수 있다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

prj1에서 구현한 read, wrtie는 stdin과 stout에서 받은 입출력에 대해서만 처리했다면, prj2에서는 file을 통한 입출력을 추가해야 한다. 그러기 위해서는 file descriptor가 필요하며, open system call에서는 file descriptor 내의 빈자리를 찾아 각각의 새로운 file에 대해 file descriptor를 통해 열 수 있게 한다. 주의해야 할 점은 이미 0번은 stdin, 1번은 stdout, 2번은 stderr로 설정되어 있다. 이외에도 filesize를 반환하는 system call, 현재의 position을 알려주는 system call, position을 변경할 수 있는 system call이 있다.

3. Synchronization in Filesystem

parent에서 여러 번 fork된 경우, child process들이 끝날 때까지 parent process는 기다려야 하는데, 이를 위해 semaphore up down을 이용한다. 또한, file이 실행 중일 때 다른 process에서 해당 file을 write하는 등의 실행을 요청하는 경우 synchronization 문제가 발생할 수 있으므로 이를 막아야한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

thread.h 파일 내에 thread 구조체에서 file descriptor 배열을 선언한다. 배열을 사용하는 이유는 각각의 파일을 배열의 요소에 연결하여 구현하면 간단하게 구현할 수 있으면서도 관리에 용이하기 때문이다. Stdin은 0번, stdout은 1번, sterr는 2번으로 고정시키고 나머지 배열의 요소에 새로 open할 파일을 할당한다.

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: 인자로 받은 파일을 크기에 맞게 filesys\_create 함수를 사용하여 생성한다. 만약 file이 null값이면 exit(-1) 예외처리 한다.

remove: create과 반대로 인자로 받은 파일을 filesys\_remove 함수를 사용하여 삭제한다.

open: 해당하는 파일을 open하고 open에 성공하면 새로운 fd를 할당하여 return한다. open하기 전과 후에 lock을 걸어 파일을 열 때 synchronization 문제가 생기지 않도록 한다.

close: 닫고자 하는 파일에 해당하는 fd를 찾아 null로 변경하여 파일을 닫는다.

filesize: 해당하는 파일의 크기를 file\_length 함수를 통해 반환한다.

seek: 해당하는 파일의 다음 편집 위치를 file\_seek 함수를 통해 변경한다.

tell: 해당하는 파일의 위치를 file\_tell 함수를 통해 반환한다.

read: project1에서 했던 키보드 입력에 대한 동작은 그대로 유지하고 파일의 내용을 읽어야 하는 경우 연결되어 있는 fd를 찾아 file\_read함수를 통해 size만큼 읽는다.

write: project1에서 stdout처리는 그대로 유지하고 파일에 쓰는 경우 해당되는 fd에 file\_write를 통해 size만큼 쓴다.

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

Semaphore을 이용하여 parent thread에서 child thread가 종료될 때까지 기다릴 수 있게 한다. 또한, child process가 적재되기 전에 parent process가 종료되어서도 안 되기 때문에 semaphore를 이용하여 load 이후까지 parent process가 기다릴 수 있도록 한다. 동시에 여러 파일에 대한 요청이 오는 경우나, 같은 파일에 대해 여러 요청이 오는 경우에 synchronization 문제가 발생할 수 있는데 lock을 사용하여 file read와 write에 처리할 수 있다. 예를 들어, 다른 요청에 의해 파일이 열려 있는데 같은 파일에 새로운 쓰기 요청이 들어온다면 이를 앞선 요청이 종료된 이후에 변경가능 하도록 해야 한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

2024.10.26 - 2024.10.27: file descriptor와 system call 구현

2024.10.28 – 2024.10.29: synchronization 구현

2024.10.30 – 2024.10.31: syn\_write /read, multi-oom test case pass 처리

2024.11.04: 보고서 작성

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

우선 file descriptor를 구현하기 위해 thread 구조체 내에 file descriptor 배열을 선언하고 stdin 0번 stdout 1번 stderr 2번으로 고정한다. Thread 구조체에는 child process의 load를 parent process가 기다리는 semaphore를 추가하고, parent를 저장할 포인터와 wait status를 추가한다. 이는 wait status에 따라 parent process가 기다리기 위함이다. File read와 write을 동시에 요청할 때 발생할 수 있는 synchronization 문제를 해결하기 위한 lock 구조체 또한 선언한다.

Syscall에는 파일을 생성하고 지울 수 있는 create, remove system call을 구현한다. 기존의 read, write system call은 키보드의 입출력만을 처리했다면 파일에 읽고 쓰는 경우는 파일에 연결되어 있는 fd를 찾아 처리하게 구현한다. filesize, seek, tell은 파일의 크기, 파일의 위치를 반환하고 변경하는 system call이다. 새로운 system call들은 모두 파일의 입출력을 구현하고 여러 요청 간의 synchronization을 방지하도록 실행전에 lock을 걸도록 구현해야 한다. 이미 실행중인 요청이 있는데 file에 쓰기 요청이 들어온 경우를 lock하기 위해서 file\_deny\_write 함수를 호출한다.

1. **연구 결과**
   1. **Flow Chart**

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

  자동 생성된 설명
* 2. System call
* 도표, 평면도, 기술 도면, 라인이(가) 표시된 사진

  자동 생성된 설명
* 3. Synchronization

텍스트, 도표, 스크린샷, 번호이(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

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

thread.h 파일에 thread 구조체 내에 file descriptor 배열과 parent process가 child process를 기다리기 위해서 필요한 wait\_status와 parent thread를 선언한다. 또한, child process가 적재하기 전에 parent process가 먼저 끝나는 것을 방지하기 위한 child\_lock semaphore가 필요하다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

thread.c 파일에 init\_thread 함수내에 parent thread를 현재 running thread로 초기화하고 file descriptor 배열을 NULL로 초기화한다. Wait\_status는 0으로 초기화한다.

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

Syscall.c 파일에 각각의 system call에 대한 처리를 switch문으로 구현한 것이다. 기존의 project1에서 사용했던 system call은 그대로 두고, 추가로 create, remove, filesize, seek. Tell에 대한 처리를 추가하였다. 각각의 system call에서 호출하는 함수들은 각각의 system call을 처리하기 위해 독립적으로 빼서 구현했다.

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

read 함수부터 살펴보면, 기존의 project1에서 구현한 stdin으로부터 read하는부분은그대로 유지하고 파일로부터 입력 받은 내용을 읽을 fd를 새롭게 할당한다. 만약에 file이 null 값이라면 exit한다. 그 후, file\_read 함수를 통해 해당하는 길이만큼 읽어서 반환한다. 중요한 점은 read하기 전에 lock을 걸고 read가 종료된 이후에 lock release를 통해 다른 요청과의 synchronization 문제가 생기지 않도록 방지해야 한다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

마찬가지로 write 함수를 살펴보면, 기존의 project1에서 구현한 stdout에 write하는 부분은 그대로 유지하고 해당하는 내용을 파일에 쓸 fd를 새롭게 할당한다. 만약에 file이 null 값이라면 exit한다. 그 후, file\_write 함수를 통해 해당하는 길이만큼 write하고 write한 내용을 반환한다. 중요한 점은 write하기 전에 lock을 걸고 write가 종료된 이후에 lock release를 통해 다른 요청과의 synchronization 문제가 생기지 않도록 방지해야 한다.

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

open 함수에서는 filesys\_open 함수를 통해 해당하는 파일을 열고 파일이 null이라면 예외처리를 하도록 구현하였다. 그 이외에는 fd 배열에서 앞에서부터 빈자리를 찾아 해당 자리에 열고자 하는 파일을 연결한다. 이 과정에서도 open하기 전에 lock을 걸고 open 이후에 lock release를 한다. 또한, 파일이 열려있을 때, 만약 같은 파일에 대해 다른 쓰기 요청이 들어온 경우 synchronization 문제가 또 생기므로 file\_deny\_write함수를 통해 write하지 못하도록 해야 한다. 또한, open하고자 하는 file의 포인터가 user가 접근할 수 있는 포인터인지를 확인하기 위해 pagedir\_get\_page 함수를 사용한다.

**텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명**

close 함수에서는 해당하는 파일을 file\_close 함수를 통해 닫고, 연결되어 있는 fd를 null로 변경한다

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

create 함수에서는 filesys\_create 함수를 통해 파일을 생성하고, remove 함수에서는 filesys\_remove 함수를 통해 파일을 지운다. File이 null이면 exit하여 예외처리한다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

마지막으로 file에 관련된 system call들로, file의 크기를 반환, file의 포인터를 반환, 포인터를 변경하는 함수를 호출한다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명**

process.c 파일에서 start\_process 함수에서 load를 실행하고 sema\_up을 통해 lock을 걸어주는데 이는 child process에서 load를 실행하기전에 parent process가 끝나는 것을 방지하기 위함이다. 걸어준 lock은 process\_execute 함수에서 sema\_down을 통해 풀어준다. 또한, child process들을 while문으로 돌면서 child의 종료를 기다리는 process\_wait함수를 반환한다.

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

마지막으로 multi-oom testcase에서 계속 오류가 발생하여 디버깅하던 도중, project1에서는 해당하는 경우가 없어서 parsing에 문제가 없었으나 입력 값 마지막에 공백으로 끝나는 경우 parsing이 제대로 되지 않는다는 문제를 발견했다. 이를 해결하기 위해, 마지막에 공백으로 끝나는 경우, 공백을 하나의 단어로 count하지 않기 위해 argc를 1 감소시키는 코드를 추가했다.

* 1. **시험 및 평가 내용**
* make check 수행 결과를 캡처하여 첨부
* 텍스트, 폰트, 흑백, 스크린샷이(가) 표시된 사진

  자동 생성된 설명
* 텍스트, 패턴, 스크린샷, 폰트이(가) 표시된 사진

  자동 생성된 설명
* 텍스트, 폰트, 스크린샷, 흑백이(가) 표시된 사진

  자동 생성된 설명