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

담당 교수 / 분반 :김영재 교수님/2분반

이름 / 학번 :20211188 방수윤

개발 기간 :10/28-11/2

1. **개발 목표**

해당 프로젝트의 개발 목표는 Pintos 운영체제의 파일 시스템에 관련된 시스템 호출을 구현하는 것이다. 구체적으로는 파일 생성(create), 삭제(remove), 열기(open), 닫기(close) 기능과 파일 크기 확인(filesize), 파일 읽기(read), 쓰기(write), 파일 위치 이동(seek), 현재 위치 확인(tell) 등의 시스템 호출을 구현하여, 사용자 프로그램이 파일 시스템에 접근하여 파일을 생성하고 삭제하며, 데이터를 읽고 쓸 수 있도록 한다. 추가적으로 file system에서 synchronization 문제를 해결해야 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

기존 Proj #1에서는 PintOS에서 파일을 다룰 때 개별 파일을 식별할 수 있는 고유한 방법이 부족하였다. 따라서 File Descriptor라는 구조를 도입하여 각 프로세스가 열어 놓은 파일들을 추적하고 관리할 수 있는 체계를 갖추는 것이 필요하다. 각 프로세스는 고유한 File Descriptor를 통해 파일을 식별하며, 일반적으로 0은 표준 입력(STDIN), 1은 표준 출력(STDOUT), 2는 표준 에러(STDERR)로 사용된다. 이 구조를 통해 파일 입출력 처리를 효율적으로 관리할 수 있다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

이전까지는 표준 입력과 출력에만 집중한 입출력 기능을 제공했다면, 이번 프로젝트에서는 보다 다양한 파일 시스템 작업을 수행할 수 있도록 여러 시스템 호출을 구현해야 한다. create, remove, open, close, filesize, read, write, seek, tell 등의 시스템 호출을 통해 파일 생성, 삭제, 열기, 닫기, 크기 확인, 읽기, 쓰기, 위치 이동 등의 기능을 제공하여 사용자 프로그램이 파일 시스템을 보다 자유롭게 조작할 수 있도록 한다. 이를 통해 입출력 및 파일 시스템의 확장된 기능을 지원하게 된다.

3. Synchronization in Filesystem

파일 시스템의 안정성을 확보하기 위해 동기화 메커니즘을 적용하는 것이 필수적이다. 여러 프로세스가 동시에 같은 파일에 접근하는 경우, 예측 불가능한 결과가 발생할 수 있기 때문에 파일 입출력 시 적절한 동기화가 필요하다. 또한 프로세스가 파일 작업을 시작하거나 종료할 때, 데이터 일관성을 보장하기 위해 락을 사용하여 공유 자원을 보호해야 한다. 이러한 동기화는 세마포어를 활용하여 동시 접근을 제어하며, 파일 시스템의 신뢰성과 일관성을 유지하도록 한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

각 프로세스가 고유한 파일 디스크립터를 관리하도록 struct file\* FD[128] 배열을 thread.h에 선언하여 사용하였다. 이 구조는 PintOS 매뉴얼에 따라 프로세스당 최대 128개의 파일을 열 수 있도록 설정되었다. 배열 형태의 자료 구조를 사용해 각 파일 디스크립터에 순차적으로 접근할 수 있으며, 이를 통해 파일을 효율적으로 관리하고 쉽게 참조할 수 있도록 하였다.

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**: 파일 이름을 받아 새로운 파일을 생성하며, 생성 성공 여부를 Boolean 값으로 반환한다.

 **REMOVE**: 주어진 파일 이름에 해당하는 파일을 삭제하고, 삭제 성공 여부를 Boolean 값으로 반환한다.

 **OPEN**: 파일 이름을 입력받아 해당 파일을 연다. 동일한 이름의 파일이 여러 번 열리지 않도록 처리하며, 파일 디스크립터 값을 반환하거나 오류 시 -1을 반환한다.

 **CLOSE**: 주어진 파일 디스크립터에 해당하는 열린 파일을 닫는다. 파일이 열려 있지 않은 경우 예외 처리를 한다.

 **FILESIZE**: 파일 디스크립터에 해당하는 열린 파일의 크기를 반환하며, 파일 접근에 실패할 경우 예외 처리를 한다.

 **READ**: 파일 디스크립터를 통해 열린 파일의 데이터를 읽어 buffer에 저장하고, 읽은 바이트 수를 반환하며 오류 시 -1을 반환한다.

 **WRITE**: 주어진 파일 디스크립터를 통해 파일에 데이터를 작성하며, 작성된 바이트 수를 반환하고 오류 시 -1을 반환한다.

 **SEEK**: 파일 포인터의 위치를 주어진 위치(pos)로 이동한다.

 **TELL**: 현재 열려 있는 파일의 포인터 위치를 반환하여, 현재 읽거나 쓰는 위치를 알려준다.

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

여러 프로세스가 동시에 파일에 접근하여 READ 및 WRITE 작업을 수행하는 경우 데이터 일관성을 보장하기 위해 **lock**을 사용하였다. 시스템 호출 초기화 단계에서 lock\_init()을 통해 락을 초기화하고, 파일 접근 전에 lock\_acquire()를 호출하여 다른 접근을 차단하며, 작업이 끝나면 lock\_release()로 락을 해제하여 다른 프로세스가 접근할 수 있게 한다.

\*\*세마포어(Semaphore)\*\*는 프로세스 간 동기화를 위해 사용된다. 자식 프로세스가 완료되기 전에 부모 프로세스가 종료되지 않도록 sema\_up()와 sema\_down()을 통해 자식 프로세스의 로드가 완료된 후에 부모 프로세스가 실행될 수 있도록 설정하여 안정적인 동작을 보장한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

10/28-10/31: file descriptor 구현, system call 구현

11/1: synchronization 구현

11/2: 보고서 작성

* 1. **개발 방법**

1. **File Descriptor**

File Descriptor는 각 스레드가 고유한 파일 디스크립터 테이블을 가질 수 있도록 struct file\* FD[128]; 배열을 thread.h에 선언하였다. PintOS 매뉴얼에 따라 프로세스당 열 수 있는 최대 파일 개수가 128개로 제한되어 있어, 각 스레드에 128개의 파일 디스크립터를 가지는 배열을 할당하였다. 배열 구조를 통해 파일 디스크립터에 순차적으로 접근할 수 있어 효율적이며, 필요한 파일을 빠르게 참조할 수 있도록 설계하였다.

1. **System Calls**

userprog/syscall.c 파일에서 create, remove, open, close, filesize, read, write, seek, tell 등 주요 파일 시스템 작업을 위한 시스템 콜을 구현하였다. 각 시스템 콜의 동작에 대해서는 II.B에서 상세히 설명하였다. 또한, userprog/exception.c 파일의 페이지 폴트 처리 조건에 not\_present 확인을 추가하여 페이지 폴트 발생 시 처리가 정확하게 이루어지도록 개선하였다.

1. **Synchronization in Filesystem**

파일 시스템 접근 시 동기화를 보장하기 위해 userprog/syscall.c에 lock\_file이라는 락을 추가하였다. 이를 통해 파일 접근 시 여러 프로세스가 동시에 동일한 파일에 접근하는 것을 방지하며, 파일을 안전하게 다룰 수 있도록 한다. 또한, threads/thread.h에 struct semaphore load\_lock을 추가하여, 자식 스레드가 완료되기 전에 부모 스레드가 종료되는 것을 막기 위해 sema\_up()와 sema\_down() 함수를 사용하여 자식 스레드의 로드가 완료된 후에 부모가 실행되도록 동기화하였다. 이를 통해 부모-자식 프로세스 간의 안정적인 작업 흐름을 보장하였다.

1. **연구 결과**
   1. **Flow Chart**

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

**텍스트, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

* **System Calls**

**텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명**

* **Synchronization in Filesystem**

**텍스트, 스크린샷, 라인, 화이트이(가) 표시된 사진

자동 생성된 설명**

* 1. **제작 내용**
* **File descripter**

[threads/thread.h]

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

struct file\* FD[128];를 thread.h에 선언하여 각 스레드마다 길이가 128인 파일 디스크립터 테이블을 가지도록 하였다. 이는 Pintos 매뉴얼에 따라 프로세스당 열 수 있는 파일의 최대 개수가 128개로 제한되어 있기 때문이다. file 구조체 배열로 구현하여, 순서대로 각 원소에 쉽게 접근할 수 있게 구성하였다.

[userprog/syscall.c]

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

File.c에 있는 파일 struct를 syscall.c에 선언해서 사용할 수 있도록 해주었다.

[threads/thread.c]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이 코드는 스레드를 초기화할 때 파일 디스크립터 배열의 각 요소를 NULL 값으로 초기화해준다. for 루프를 사용하여 FD[128] 배열의 모든 인덱스를 순회하면서 NULL로 설정하고, 이후에는 세마포어들을 0으로 초기화하여 동기화를 설정한다. 이를 통해 각 스레드가 독립적인 파일 디스크립터와 동기화 변수를 가지고 초기화되도록 구성하였다.

* **System calls**

[userprog/exception.c]

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

not\_present 조건을 추가함으로써 메모리에 없는 페이지에만 필요한 처리를 수행하여, 메모리 관리와 접근 제어에 효율성을 높인다. not\_present는 오류 코드의 PF\_P 비트를 확인하여 설정하며, 이는 페이지가 메모리에 존재하지 않는 상황을 의미한다. 시스템 호출에서 not\_present 조건을 통해 필요한 페이지를 올바르게 로드하거나 접근을 제한하는 기능을 구현할 수 있도록 한다.

[userprog/syscall.c]

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

syscall\_handler() 함수의 Syscall switch 문에 이번 프로젝트에서 구현한 시스템 호출인 create, remove, open, close, filesize, read, write, seek, tell을 추가하였다. Check\_addr(함수를 사용하여 메모리 접근에 대한 유효성 검사를 수행하며, 올바르지 않은 주소 접근을 방지하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

CREATE 함수는 파일의 이름과 크기를 인자로 받아, 지정된 파일을 생성하도록 한다. 파일 이름이 NULL인 경우, EXIT(-1)을 호출하여 예외 처리를 수행하며, 정상적인 경우에는 filesys\_create() 함수를 이용하여 파일을 생성한다. filesys\_create()의 반환 값을 success에 저장한 뒤, 생성 성공 여부를 반환하도록 한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

파일의 이름을 인자로 받아, 지정된 파일을 삭제하도록 한다. 파일 이름이 NULL인 경우, EXIT(-1)을 호출하여 예외 처리를 수행한다. 정상적인 경우에는 filesys\_remove() 함수를 이용해 파일을 삭제하며, filesys\_remove()의 반환 값을 success에 저장한 뒤, 삭제 성공 여부를 반환하도록 한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

파일 디스크립터 fd를 인자로 받아 해당 파일의 크기를 반환하도록 한다. fd가 0 이하이거나 NULL인 경우, EXIT(-1)을 호출하여 오류를 처리한다. 정상적인 경우, file\_length() 함수를 사용하여 현재 스레드의 파일 디스크립터 테이블에서 해당 파일의 길이를 구하고, 이를 length 변수에 저장한 후 반환한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

파일 이름을 인자로 받아 filesys\_open()으로 파일을 연다. 파일을 열기 전, 파일 이름에 대한 유효성 검사를 수행한다

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

file\_seek()을 통해 인자로 받은 파일 디스크립터 값에 해당하는 파일 포인터의 위치를 지정된 position으로 변경한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

READ() 함수는 주어진 파일 디스크립터 값을 확인한 후, 해당 파일에서 지정된 크기만큼 데이터를 읽어 buffer에 저장한다. fd가 0일 경우 표준 입력에서 읽어오고, 2보다 큰 경우 파일에서 데이터를 읽는다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

TELL() 함수는 주어진 파일 디스크립터 값의 현재 파일 위치를 반환한다. 파일 디스크립터가 유효하지 않은 경우, EXIT(-1)을 호출하여 종료한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

CLOSE() 함수는 주어진 파일 디스크립터 값에 해당하는 파일을 닫는다. 파일 디스크립터가 유효하지 않은 경우 EXIT(-1)을 호출하여 종료한다. 파일을 닫은 후, 해당 파일 디스크립터를 NULL로 설정하여 더 이상 접근할 수 없도록 한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

WRITE() 함수는 파일 디스크립터와 버퍼, 쓰기 크기를 인자로 받아 해당 파일에 데이터를 쓴다. 만약 fd가 1인 경우, 표준 출력(STDOUT)에 데이터를 출력하며, 그 외의 유효한 파일 디스크립터에 대해서는 파일에 데이터를 쓰고, 성공적으로 쓴 바이트 수를 반환한다. buffer가 NULL이거나 유효하지 않은 파일 디스크립터인 경우, EXIT(-1)을 호출하여 오류를 처리한다.

* **Synchronization in Filesystem**

[userprog/syscall.c]

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

OPEN() 함수는 파일을 열 때 동시 접근을 막기 위해 lock을 활용하여 동기화를 구현하였다. Syscall\_init에서 lock\_init(&locking)으로 환경을 설정한 후, lock\_acquire(&locking)을 통해 파일 접근 시 락을 걸고, 작업이 끝난 후 lock\_release(&locking)으로 해제하여 다른 접근을 허용하였다. 이는 READ, WRITE에서도 동일하게 적용하여 파일 시스템 호출 중 충돌을 방지하였다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[userprog/process.c]

MAKE CHECK시에 3개 항목이 FAIL이 나와서 process.c의 내용을 수정하였다. process\_execute 함수와 start\_process 함수는 새로운 프로세스를 생성하고 실행하는 역할을 수행한다.

process\_execute 함수는 전달받은 파일 이름을 기반으로 새로운 스레드를 생성하며, 이를 통해 사용자 프로그램을 실행할 수 있는 환경을 설정한다. 먼저 파일 이름을 복사하여 이후의 작업에서 사용할 수 있도록 하고, 파일이 존재하는지 확인한 후 thread\_create를 호출하여 스레드를 생성한다. 이후 자식 스레드가 로드 완료 신호를 보내기 전까지 부모 스레드는 대기하도록 sema\_down을 사용하여 동기화를 구현한다.

start\_process 함수는 새롭게 생성된 스레드에서 실행되며, 전달받은 파일 이름을 사용하여 프로그램을 로드한다. 인터럽트 프레임을 초기화하고 load 함수를 통해 ELF 실행 파일을 메모리에 로드한다. 로드 성공 여부를 load\_success에 저장한 후 sema\_up을 통해 부모 스레드에게 로드 완료를 알린다. 로드가 실패하면 현재 스레드를 종료하고, 성공하면 사용자 프로그램을 실행하기 위해 인터럽트 처리 구조로 진입한다.

이 코드는 sema\_down과 sema\_up을 사용하여 프로세스 생성 과정에서 부모와 자식 스레드 간의 동기화를 구현했으며, 자식 스레드가 완전히 로드될 때까지 부모 스레드가 대기하도록 설계되었다.

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

[putty에서 성공 화면]

* 스크린샷, 패턴, 타이포그래피, 예술이(가) 표시된 사진

  자동 생성된 설명
* 텍스트, 스크린샷이(가) 표시된 사진

  자동 생성된 설명

[전체화면]

* 텍스트, 스크린샷, 타이포그래피, 디자인이(가) 표시된 사진

  자동 생성된 설명