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

담당 교수 / 분반 : 박성용 교수님 / 1분반

이름 / 학번 : 전용본 /. 20181683

개발 기간 :

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

프로젝트1에서 구현했던 System call에서 구현하지 않았던 File system과 관련된 System call들을 추가로 구현하고 read, write System call은 최소한으로만 작동했기에 이를 보완한다. 또 File system을 접근/관리할 때 발생하는 동기화 문제를 해결해야 한다. 이를 통해 File system을 user가 system call을 통해 관리할 수 있게 되는 것이 목표이다.

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 구현할 경우 OS의 process들은 각자 고유의 File descriptor table을 가질 수 있다. Process들은 이를 이용해 File system에 접근, 관리할 수 있다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

프로젝트 1에서 구현하지 않았던 create, remove, open, close, filesize, seek, tell System call을 구현한다. 기존의 read, write System call은 표준 출력, 표준 입력에 한해서만 작동했기에 이를 보완/개선하며 동기화 문제를 해결한다면 File system에 대한 System call이 모두 구현되므로 File system을 관리할 수 있게 된다.

3. Synchronization in Filesystem

File system의 동기화는 File에 read/write을 할 때 여러 process가 동시에 접근할 경우 원하지 않는 결과값이 나오는 것을 방지하기 위해 구현한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

pintOS에서 PCB는 thread.h에 구현된 struct thread이다. 각 thread가 file descriptor를 관리할 수 있도록 struct thread에 file descriptor 배열을 선언한다. pintOS에서 thread가 관리할 수 있는 file descriptor의 개수는 128개로 제한되어 있고 크지 않은 상수값이기에 배열로 선언한다면 관리가 굉장히 용이하기에 선택했다.

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 : 새로운 File을 생성하는 system call이다. File 이름이 제대로 주어지지 않는다면 종료한다.

remove : 기존의 File을 제거하는 system call이다. File 이름이 제대로 주어지지 않는다면 종료한다.

open : 인자로 주어진 File이 존재한다면 이를 사용할 수 있도록 하고 process의 file descriptor table에 추가하는 system call이다.

close : process의 file descriptor table에서 관리하고 있던 file을 table에서 삭제하고 이를 사용하지 못하게 하는 system call이다.

filesize : File의 크기를 반환하는 system call이다.

Seek : File을 읽기 시작하는 위치를 조정하는 system call이다.

tell : File을 읽기 시작하는 위치를 반환하는 system call이다.

read : File의 내용을 원하는 만큼 buffer에 저장해 반환하는 system call이다. 읽은 buffer의 크기 만큼을 반환한다.

write : File에 buffer에 저장된 내용을 작성하는 system call이다. 작성한 buffer의 크기 만큼을 반환한다.

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

여러 process가 동시에 같은 파일에 접근해 행동을 수행할 경우 서로 순서에 따라 원하지 않는 결과값을 얻을 수 있기에 이를 방지하고자 Lock을 사용해 동기화 문제를 해결한다. 파일에 접근하는 system call의 경우 critical section을 lock\_acquire, lock\_release 로 감싸서 파일이 수정되지 못하게 한다.

Process간의 동기화 문제를 처리하기 위해 Semaphore을 사용한다. 부모 process가 많은 자식 process를 생성할 때 모든 자식들이 load가 될 수 있게 기다려야 한다. 프로젝트1에서 사용되지 않은 새로운 semaphore을 만들어 모든 자식이 load가 될 수 있게 한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

10/6 ~ 10/7 : File descriptor 구현 및 전체적인 flow 파악

10/8 ~ 10/10 : create, remove, open, close, filesize, seek, tell 구현

10/11 ~ 10/14 : Synchronization 문제 해결

10/15 ~ 10/16 : Test 결과를 토대로 debugging

10/17 ~ 10/18 : 보고서 작성

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

1. File descriptor

pintOS의 PCB에 해당하는 threads/thread.h에 있는 struct thread에 file descriptor array를 추가해야 한다. Init\_thread에서 이를 NULL로 초기화하고 다음에 구현할 System call들에서 적절하게 사용한다.

2. System calls

Thread.h에 구현한 file descriptor를 이용해서 file system을 관리/접근하는 system call인 create, remove, open, close, read, write, tell, seek, filesize를 구현한다. 각각을 구현할 때 filesys/file.c와 filesys/filesys.c의 적절한 함수들을 사용한다.

3. Synchronization in File system

동기화 문제를 해결하기 위해 syscall.c에 lock 변수를 선언하고 syscall\_init에서 이를 초기화한다. syscall.c에서 file에 변화를 주는 system call들을 사용할 경우 critical section에서 lock을 acquire하고 release하는 행동을 통해 동기화 문제를 해결한다. .

1. **연구 결과**
   1. **Flow Chart**

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

1. File descriptor

2. System calls

3. Synchronization in File system

* 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

1) threads/thread.h



Struct thread 구조체 안에 file\_descriptor table을 배열로 선언한다. pintOS에서 각 thread가 관리할 수 있는 file descriptor는 128개 이므로 배열의 길이는 128로 선언했다.

2) threads/thread.c

텍스트이(가) 표시된 사진

자동 생성된 설명

Thread 생성시에 초기화하는 함수인 init\_thread 함수 내부에 thread의 file descriptor들을 초기화하였다.

3)

2. System calls & synchronization

1) userprog/syscall.c

1-1) lock 변수 선언



System call 구현 시에 사용할 lock 변수를 선언한다.

1-2) lock 변수 초기화

텍스트이(가) 표시된 사진

자동 생성된 설명

Lock\_init 함수를 이용해 lock 변수를 초기화하였다.

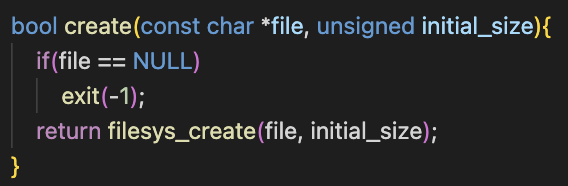
1-3) system call handler 수정

텍스트이(가) 표시된 사진

자동 생성된 설명

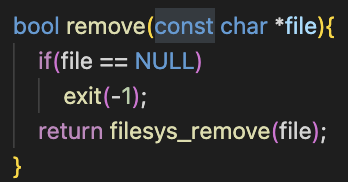
프로젝트 1에서 구현했던 syscall\_handler 함수 내부에 새로 구현할 system call들을 syscall number에 맞게 적절한 함수를 호출할 수 있게 수정했다.

1-4) create system call 구현



인자로 생성할 file 이름과 초기 size를 받아 filesys\_create 함수를 이용해 file을 생성하게 구현했다. 만약 file 이름이 NULL인 경우 exit(-1) 함수를 이용해 종료했다.

1-5) remove system call 구현



인자로 제거할 file 이름을 받아 filesys\_remove 함수를 이용해 file을 삭제할 수 있게 구현했다. 만약 file 이름이 NULL인 경우 exit(-1) 함수를 이용해 종료했다.

1-6) open system call 구현

텍스트이(가) 표시된 사진

자동 생성된 설명

인자로 열고자 하는 file 이름을 받아 filesys\_open 함수를 이용해 열 수 있게 했다. 만약 file 이름이 NULL인 경우 exit(-1) 함수를 이용해 종료되게 하였다. File system에 직접적으로 접근하는 system call이므로 lock을 이용해 다른 process가 수정하지 못하게 했다. 또 pintOS에서는 현재 실행되고 있는 executable file이 수정되는 것을 허용하지 않으므로 file\_deny\_write 함수를 이용해 수정을 방지했다. 현재 thread의 file descriptor table을 순회하면서 비어 있는 배열 중에 가장 앞에 file descriptor를 저장했다. File descriptor table에 저장한 index를 반환한다.

1-7) filesize system call 구현

텍스트이(가) 표시된 사진

자동 생성된 설명

인자로 file descriptor number를 받아 file\_length 함수를 이용해 파일의 길이를 구할 수 있게 하였다. 만약 해당 file이 open되어 있지 않다면 exit(-1) 함수를 이용해 종료되게 하였다.

1-8) seek system call 구현

텍스트이(가) 표시된 사진

자동 생성된 설명

인자로 file descriptor number와 수정을 원하는 position을 받아 file\_seek 함수를 이용해 file의 position을 수정할 수 있게 하였다. 만약 해당 file이 open되어 있지 않다면 exit(-1) 함수를 이용해 종료되게 하였다.

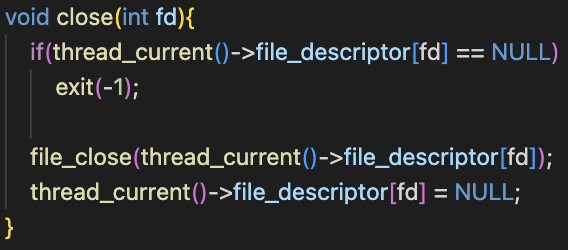
1-9) tell system call 구현

텍스트이(가) 표시된 사진

자동 생성된 설명

인자로 file descriptor number를 받아 file\_tell 함수를 이용해 해당 file의 현재 position을 구할 수 있게 하였다. 만약 해당 file이 open되어 있지 않다면 exit(-1) 함수를 이용해 종료되게 하였다.

1-10) close system call 구현



인자로 file descriptor number를 받아 file\_close 함수를 이용해 해당 file이 종료될 수 있게 하였다. 종료가 되었기에 해당 thread가 관리하고 있는 file descriptor table에서도 삭제하였다. 만약 해당 file이 open되어 있지 않다면 exit(-1) 함수를 이용해 종료되게 하였다.

1-11) exit system call 수정

텍스트이(가) 표시된 사진

자동 생성된 설명

기존의 exit system call에서 file system과 synchronization 관련된 부분을 추가 구현하였다. Process가 종료될 때 메모리 누수를 방지하기 위해 관리하고 있는 file descriptor table에서 열려 있는 file descriptor를 모두 close한다.

1-12) read system call 수정

텍스트이(가) 표시된 사진

자동 생성된 설명

프로젝트1에서 구현했던 read system call을 표준 입력말고 일반적인 상황에서도 사용할 수 있게 하고 synchronization 부분을 추가 구현했다. 인자로 받은 descriptor number가 표준 입력이 아닌 경우에 해당 file이 열려 있지 않은 경우에는 exit(-1)을 이용해 종료되게 하였다. File descriptor table을 이용해 file 변수를 받은 후에 이를 이용해 file\_read 함수를 이용해 file의 내용을 읽어 들일 수 있게 하였다. File에 직접적인 접근/수정을 하는 함수이므로 critical section에 lock을 걸어 수정을 방지하였다. File에서 읽어들인 buffer의 크기만큼 반환한다.

1-13) write system call 수정

텍스트이(가) 표시된 사진

자동 생성된 설명

프로젝트1에서 구현했던 write system call을 표준 출력말고 일반적인 상황에서도 사용할 수 있게 하고 synchronization 부분을 추가 구현했다. 인자로 받은 descriptor number가 표준 출력이 아닌 경우에 해당 file이 열려 있지 않은 경우에는 exit(-1)을 이용해 종료되게 하였다. File descriptor table을 이용해 file 변수를 받은 후에 먼저 해당 file이 deny\_write 변수가 true인지 확인해서 수정을 방지한다. 그 다음에 file\_read 함수를 이용해 file에 작성할 수 있게 하였다. File에 직접적인 접근/수정을 하는 함수이므로 critical section에 lock을 걸어 수정을 방지하였다. File에 작성한 buffer의 크기만큼 반환한다.

3. Synchronization in File system

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