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

담당 교수 / 분반 : 김영재, 2분반

이름 / 학번 : 곽우철, 20161564

개발 기간 : ~10/16

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

project 1에서 구현하지 않은 file system call 관련 함수들 구현하고(create, remove, filesize 등등) 여러 상황들(동기화 문제, memory 누수 문제)을 처리한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

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

1. File Descriptor

각 process마다 다른 file descriptor를 가지고 있으므로 이를 thread의 요소에 추가하고 사용한다(사이즈 128).

2. (이번 프로젝트에서 구현해야 하는) System Calls

create, remove는 파일을 생성하고 제거하는데 쓰이며 open, close는 열고 닫을 때, filesize는 파일의 사이즈를, read, write는 파일을 읽고 쓰는데 쓰이며 seek, tell과 같이 파일 포인터 관련 함수도 있다. 즉, 파일을 통해 수행 가능한 여러 작업들을 원활하게 이뤄지기 하기 위해서 이런 system call들을 구현해야 한다.

3. Synchronization in Filesystem

같은 파일에 write하면서 동시에 read하는 일이 생기면 곤란하므로 semaphore를 이용해서 이를 방지하고 동기화가 이루어지도록 한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

filesys.c에 있는 file구조체를 이용한다. 이것이 thread와 같이 사용하기도 편리하고 file descriptor number들도 관리하기 용이하다. thread.h에서 struct file\*fd[FD\_LEN];와 같이 선언하고 사용한다.

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)

create의 경우 file 이름, initial size가 넘어오면 파일을 생성한다. remove의 경우 이와 반대로 file 이름이 넘어오면 삭제한다. open은 file 이름이 넘어오면 이를 열고 file descriptor를 만들어서 현재 thread의 것으로 만들어준다. filesize 함수는 파일의 사이즈를 반환하고(fd number가 넘어오면 이것을 이용해서 file descriptor에 접근) close는 마찬가지로 fd가 넘어오면 이를 이용해 파일을 닫아준다. read 함수는 매개변수로 온 size만큼 file에서 읽고 이를 buffer로 저장하고 write의 경우는 반대로 size만큼 buffer에서 읽어서 file에 저장하는 것이다. 두 개 다 stdin, stdout의 경우를 조건문으로 나눠서 고려한다(file이 아닌 stdin stdout으로 입출력하는 것이므로). seek 함수는 file에서 포인터 위치, 즉 파일에서 다음으로 읽고 쓸 위치를 매개변수 position을 이용해 지정하는 것이고 tell은 이 위치를 반환해주는 것이다.

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

먼저 critical section들에 대한 처리를 해주는데 이는 구현한 file system call 함수들 중에 read, write, open 부분이다. 각각에 대해 lock이나 semaphore를 함수의 앞뒤로 걸어주면 되는데 먼저 syscall.c의 앞부분에 struct lock lock1; struct semaphore rd; 와 같이 선언을 해주고 syscall\_init에서 초기화 하면 된다. lock의 경우 각각의 함수 앞뒤에 lock\_acquire 그리고 lock\_release를 해주면 다른 process가 접근을 하지 못한다. semaphore의 경우는 sema\_down과 sema\_up을 활용해서 동이랗게 하면 된다. 실행중인 파일에 대한 접근은 open 함수 안에서 thread이름과 file이름을 비교하고 file\_deny\_write함수를 불러서 방지하면 된다. 또한 process.c에서 execute부분에서 thread를 생성하는데 여기서 parent thread가 execute이 너무 빨리 끝나서 child thread의 load 실패를 무시하게 되는 경우가 있는데 이것을 semaphore를 이용해서 잘 막아야한다. wait하는 과정과 exit하는 것(parent가 먼저 exit되면 안되는)은 proj1에서 동기화 구현이 완료됐다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

주어진 단계대로 먼저 create, remove등 file 관련 system call들을 filesys.c에 있는 함수들을 이용하여 구현한다. 다음 세션에서 critical section을 찾아서 semaphore를 건다. 그 후에 make check에서 요구하는 test 조건들을 확인 후 부족한 부분을 수행한다..

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

system call 관련해서는 전부 syscall.c에서 이루어진다. thread관련 요소들을 추가하고 초기화하는 작업은 threads.h, threads.c에서 이루어지고 동기화 작업, 예외처리 작업은 process.c와 exception.c도 수정해야한다.

* + 수정하거나 추가해야 하는 자료구조

thread.h에서 thread의 요소로 struct file\*fd[FD\_LEN]; 배열을 추가, proj1에서도 사용됐던 thread의 요소인 childs는 child list도 사용된다. 이 외 따로 queue나 stack등의 자료구조를 이용하는 경우는 없음.

* + 수정하거나 추가해야 하는 함수

각각의 system call 함수들은 따로따로 함수로 존재한다. 이것들을 구현하는데 추가로 사용하는 함수는 없다. 거의 다 있는 함수들을 가져다 쓰는 작업이다.

1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 3가지 항목에 대하여 Flow Chart 작성
  1. **제작 내용**
* II. B. 개발 내용의 각 3가지 항목에 대하여 실직적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)

fd의 경우는 앞에서 말한 것과 같이 thread.h에서 struct file\*fd[FD\_LEN];를 선언해서 thread와 연결되게 해주고 syscall.c의 open 함수에서 if(thread\_current()->fd[i]==NULL)일때 즉 비어있을 때 thread\_current()->fd[i]=fp로 한다. (fp는 filesys\_open을 통해 연 file 구조체) exit 함수에서 이 fd들이 초기화 된다.

각각의 system call들을 간단히 살펴보면 create은 filesys\_create() 이용, reove는 filesys\_remove() 이용, open은 filesys\_open() 이용, filesize는 file\_length()함수 이용, read, write, tell, seek 함수도 마찬가지로 filesys.c의 함수를 그대로 이용해서 반환값만 맞춰주면 된다.

동기화를 위해 read, write, open에 lock과 semaphore를 걸어주는 것은 앞에서 말했으므로 생략한다. 그냥 함수의 앞 뒤에 lock\_acquire, lock\_release, sema\_up, sema\_down을 추가해주면 되는 것이다. 또, parent와 child의 load 문제를 위해서는 thread.h에 struct thread\*parent, struct semaphore exec\_lock, child의 load 성공여부를 알려주는 lflag를 추가한다. 그리고 process.c에서 execute 함수에서 thread\_create를 하고 바로 아래에 sema\_down(&thread\_current()->exec\_lock);을 해서 막아주고 child의 load가 끝난 시점인 start함수의 if(!success) 위에 sema\_up(&thread\_current()->parent->exec\_lock);를 해서 풀어준다. 또한 이때 load 성공 여부를 if(!success)안에 (thread\_current())->lflag=true; (load 실패가 true)와 같이 전달하고 이를 execute에서 parent가 child list를 통해 확인하고 load 실패인 것을 확인시에 process\_wait를 해서 예외처리를 해준다.

* 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명

proj1에서 사용한 valid\_check 말고는 새로 정의한 것은 없다.

* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명

test중 다음3개의 문제가 잘 안 풀렸는데 뒤의 2개는 lock이나 semaphore의 위치도 조정해보고 동기화의 문제를 찾아봤는데 안 풀려서 multi-oom을 먼저 해결하니 뒤의 2개가 같이 해결되었다.

FAIL tests/userprog/no-vm/multi-oom

FAIL tests/filesys/base/syn-read

FAIL tests/filesys/base/syn-write

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

다음 페이지에 첨부

* 텍스트이(가) 표시된 사진

  자동 생성된 설명
* make check 입력시
* 아래 페이지에 결과들 있음
* 텍스트이(가) 표시된 사진

  자동 생성된 설명
* 텍스트이(가) 표시된 사진

  자동 생성된 설명