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

담당 교수 / 분반 : 문의현 / 2분반

이름 / 학번 : 20180492 / 김범준

개발 기간 : 2022. 11. 3 ~ 2022. 11. 14

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

PintOS 프로젝트 2에서는 파일과 관련된 시스템 콜을 구현하는 것을 목표로 한다. 구체적으로, 프로젝트 1에서 구현하지 않았던 시스템 콜 중 create, remove, open, close, filesize, read, write, seek, tell을 구현하고 프로젝트 1에서 구현했던 read와 write 시스템 콜의 경우 stdin이 아닌 경우에서 정상적으로 작동하도록 구현한다. 추가적으로, 파일을 읽고 쓸 때의 동기화 또한 구현한다.

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 프로세스가 실행될 때, 해당 프로세스가 어떠한 파일에 접근하려고 할 때 사용하는 파일의 번호이다. 파일이 저장될 때 메모리의 남은 공간에 연속적 할당이 된다. 이후에 어떠한 프로세스가 해당 파일에 접근을 하려고 할 경우, 이미 메모리에 할당이 되었던 파일에 file descriptor를 통하여 파일 접근이 가능하다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

프로젝트 1에서 구현되어 있는 유저 스택에 접근하여 명령어를 전달받아, 그것을 커널과 연결시키는 시스템 콜 핸들러를 구현한다. 시스템 콜 핸들러의 기본적인 틀은 프로젝트 1에서 짜여져 있으므로, 새로운 명령어에 대한 시스템 콜을 추가적으로 구현한다. 또한, 프로젝트 1에서 stdin의 경우만 구현했던 read와 write 시스템 콜을 file descriptor를 이용하여 파일의 읽기와 쓰기에 대한 경우도 구현을 한다. 최종적으로, 파일에 관한 시스템 콜을 구현하는 것을 목적으로 한다.

3. Synchronization in Filesystem

프로세스들이 파일을 읽거나 파일에 어떤 내용을 쓰게 될 경우, 여러 프로세스가 한 번에 해당 파일에 접근할 수 없도록 관리하는 동기화가 필요하다. 또한, 이미 실행되어 읽어들여지고 있는 파일이 종료되지 않도록 보호하는 동기화도 필요하다. 프로젝트 2에서는 해당 동기화를 lock 구조체와 semaphore 구조체를 활용하여 구현한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

file descriptor 구현에는 file 구조체의 배열 자료구조를 사용한다. 유저 스택에서 file descriptor 값에 해당하는 인덱스를 넘겨받아, 해당 인덱스를 이용하여 파일에 직접적으로 접근할 수 있기 때문이다.

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 : 새롭게 생성할 파일의 이름과 크기를 입력받고, 해당 이름과 크기를 가지는 파일을 생성한다.

-remove : 삭제할 파일의 이름을 입력받고, 해당 이름을 가지는 파일을 찾아서 삭제한다.

-open : 접근하여 열 파일의 이름을 입력받고, 해당 이름을 가지는 파일을 찾아서 open한다. 이때, file descriptor를 참조하여 파일에 접근하게 된다.

-close : 접근하여 닫을 파일의 이름을 입력받고, 해당 이름을 가지는 파일을 찾아서 close한다.

-filesize : file descriptor 값을 참조하여 해당 file descriptor의 파일의 크기를 반환한다.

-seek : file descriptor 값을 참조하여, 해당 파일에서 읽는 위치를 다음 byte로 옮긴다.

-tell : file descriptor 값을 참조하여, 해당 파일에서 읽는 위치를 반환한다.

-read : stdin이 아닌 입력에 대하여 file descriptor 값을 참조하여 해당 값의 파일에 접근하여 정해진 길이만큼 읽어들인다.

-write : stdin이 아닌 입력에 대하여 file descriptor 값을 참조하여 해당 값의 파일에 접근하여 정해진 길이만큼 쓴다.

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

-lock : 어떤 프로세스가 파일을 열거나 읽거나 쓸 때 다른 프로세스가 해당 파일에 접근할 수 없게 하기 위해 구조체 lock을 사용한다. 열기, 읽기, 쓰기 작업을 할 때, 해당 작업을 시작하기 전에 lock\_acquire를 통해 lock을 실행시켜 다른 프로세스의 접근을 막고, 끝나고 난 뒤에는 lock\_release를 통해 lock을 풀어주어 다른 프로세스가 접근할 수 있도록 한다.

-semaphore : 프로젝트 1에서 나아가, 어떤 프로세스가 실행될 때의 동기화를 구현한다. 새로운 세마포어를 선언하고, process\_execute에서 sema\_down, start\_process에서 sema\_up을 통해 세마포어를 관리해주었다. 추가적으로, start\_process에서 어떤 프로세스가 load에 실패하였을 경우 쓰레드의 err\_flag에 true를 할당하고 프로세스를 종료시키는 코드를 추가하였다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

2022. 11. 3 ~ 2022. 11. 4 : file descriptor 및 create 시스템 콜 구현

2022. 11. 5 ~ 2022. 11. 9 : remove, open, close, filesize, seek, tell, read, write 시스템 콜 구현

2022. 11. 10 ~ 2022. 11. 14 : 동기화 구현

2022. 11. 14 : 보고서 작성

* 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

먼저, file 구조체를 userprog 및 threads 디렉토리의 소스 코드에서 사용할 수 있도록 file.c 파일에서 file.h 파일로 옮긴 뒤, thread.h, syscall.c, process.c에서 include한다. 다음으로, thread.h의 thread 구조체에서 file 구조체 배열 fd를 선언한다. 이때, file descriptor array의 크기는 핀토스 매뉴얼을 참조하여, 128로 선언하였다. 또한 thread.c의 init\_thread 함수에 file descriptor array를 초기화하는 코드를 추가하였다.

2. system call

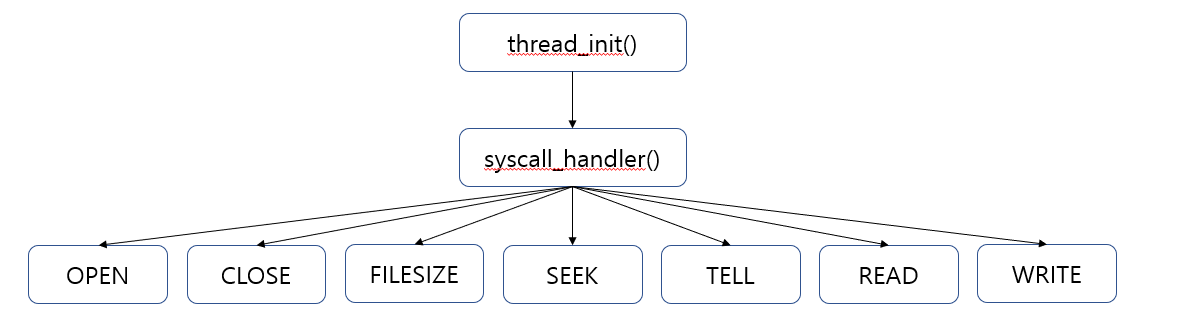
새롭게 구현할 시스템 콜 create, remove, open, close, filesize, seek, tell 시스템 콜을 syscall.c의 switch문에 추가하여준다. 해당 분기에서는 file descriptor값을 유저 스택으로부터 넘겨받아 파일을 참조하게 된다. 추가적으로, 프로젝트 1에서 구현하였던 sysexit함수에 프로세스가 종료될 때 fd배열에 남아있는 파일을 닫고, 모든 인덱스를 NULL로 초기화하는 코드를 추가하였다.

3. synchronization

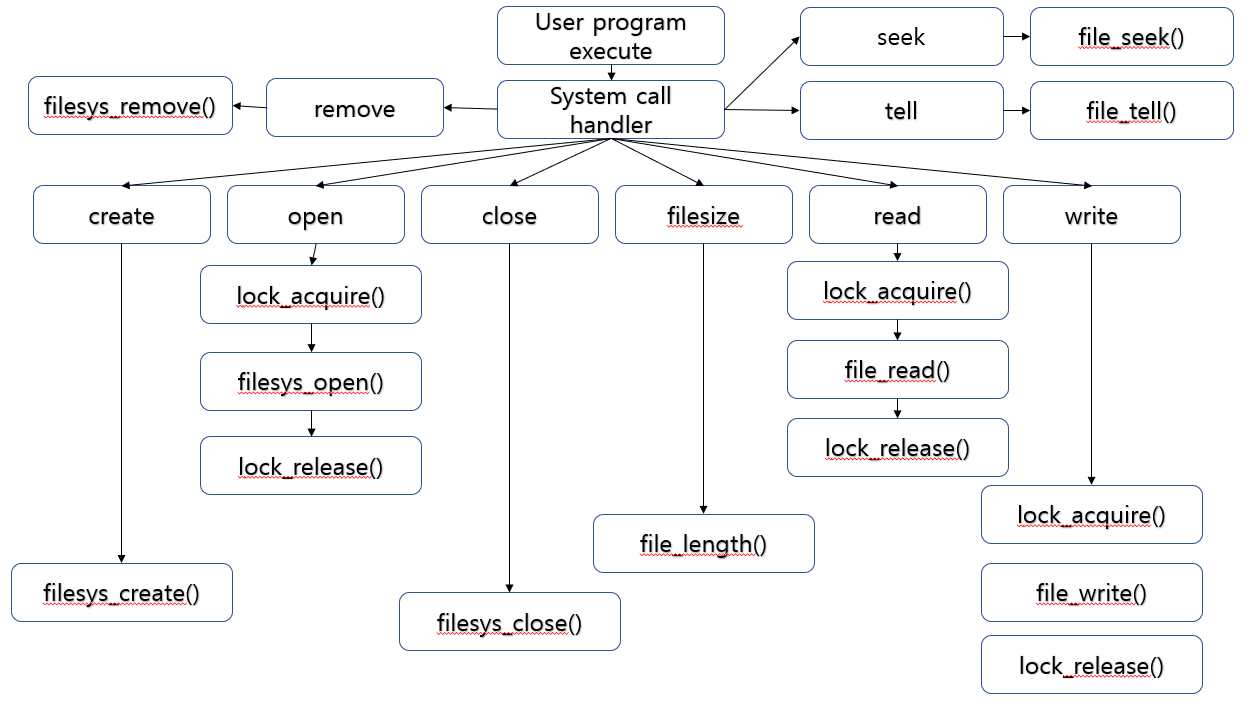
동기화를 위하여 사용할 lock 구조체 flock을 thread.h에 선언하고, thread\_init 함수에서 초기화한다. 또한, 새로운 세마포어 L을 thread 구조체에 선언하고, init thread에서 초기화한다. 추가적으로, 프로세스 실행시 load에 실패하였을 경우 exit 시스템 콜을 호출하기 위한 플래그 변수 err\_flag를 추가하고, init\_thread에서 초기화한다.

1. **연구 결과**
   1. **Flow Chart**

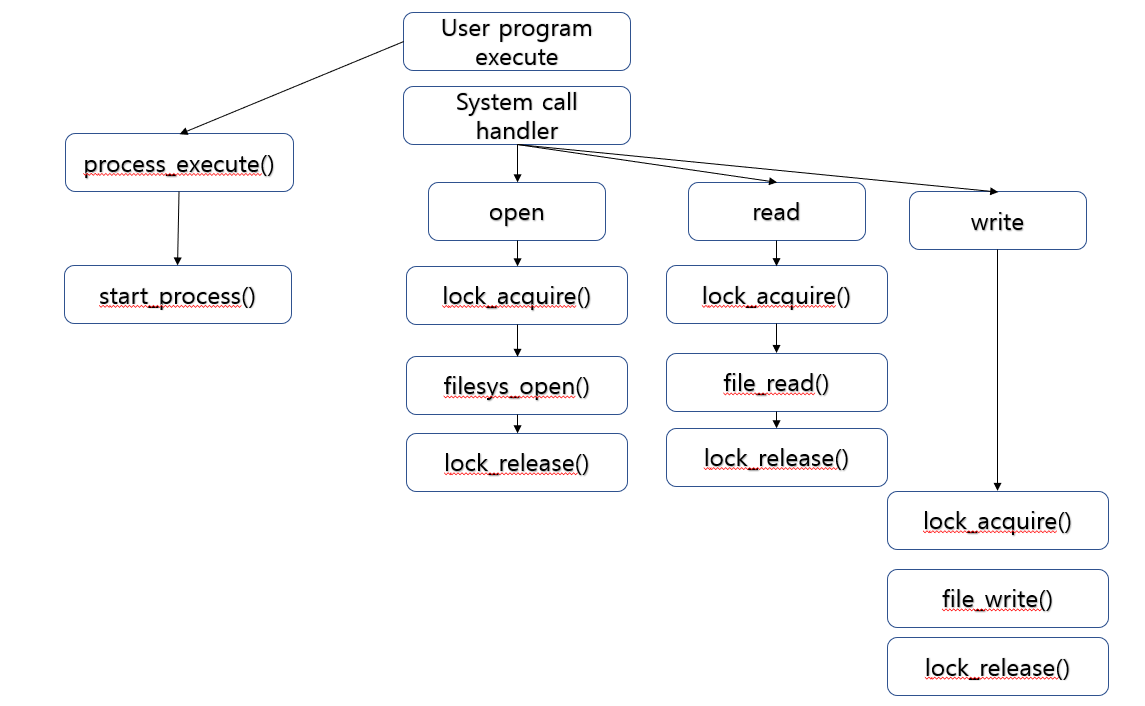
* II. B. 개발 내용의 각 3가지 항목에 대하여 Flow Chart 작성
  + 1. **file description**

****

**2. system call**

****

**3. synchronization**

****

* 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. thread.h



텍스트이(가) 표시된 사진

자동 생성된 설명

open, read, write에서 동기화를 위한 lock 구조체를 사용하기 위해 flock을 선언하였다. 추가적으로 thread 구조체에 file descriptor array, 해당 쓰레드의 부모 쓰레드를 가리키는 thread 구조체 parent, 새로운 세마포어 L, err\_flag를 선언하였다.

2. thread.c



텍스트이(가) 표시된 사진

자동 생성된 설명

thread.h에서 선언한 flock을 thread\_init 함수에서 초기화하였다. 또한, init\_thread 함수에서 file descriptor array, 세마포어 L, parent, err\_flag를 초기화하였다.

3. file.c 및 file.h

텍스트이(가) 표시된 사진

자동 생성된 설명

file 구조체를 다른 디렉토리에서 사용하기 위해 file.c의 file 구조체 선언을 file.h에 새로 선언하였다.

4. syscall.c

텍스트이(가) 표시된 사진

자동 생성된 설명

SYS\_CREATE에서는 먼저 유저 스택에서 받아온 메모리에 불법적인 접근일 일어나는지 체크하여준다. 다음으로 생성하고자 하는 파일의 이름에 해당하는 문자열이 null일 때 sysexit(-1)을 호출하고, 정상적으로 인자를 넘겨받았을 때, 인자에 해당하는 이름과 크기를 가지는 파일을 생성한다.

텍스트, 실내이(가) 표시된 사진

자동 생성된 설명

유저 스택에서 넘겨받은 메모리에 불법적인 접근이 일어나는지 확인하고, 삭제하고자 하는 파일의 이름에 해당하는 문자열이 비어있다면 sysexit(-1)을 호출한다. 그렇지 않다면 인자로 넘겨받은 이름에 해당하는 파일을 삭제한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

유저 스택에서 넘겨받은 메모리에 불법적인 접근이 일어나는지 검사하고, 열려는 파일의 이름이 비어있을 경우 sysexit(-1)을 호출한다. 그렇지 않을 경우, lock을 걸고 파일을 연 뒤, 현재 실행중인 프로세스의 file descriptor array의 빈 곳에 새로 연 파일을 할당한다. 마지막으로, lock을 해제한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

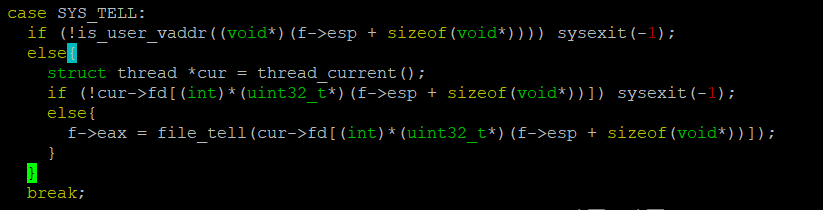
먼저 유저 스택에서 넘겨받은 메모리에 불법적인 접근이 일어나는지 검사한다. 그렇지 않으면, 인자로 넘겨받은 file descriptor값에 해당하는 파일이 있는지 검사한 다음, 해당 파일을 닫고 해당하는 file descriptor에 null을 할당한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

SYS\_FILESYS에서는 유저 스택에서 넘겨받은 메모리에 불법적인 접근이 일어나는지 검사한다. 다음으로, 넘겨받은 file descriptor 값에 해당하는 파일이 있는지 검사하고, 해당 파일의 길이를 반환한다.

SYS\_SEEK에서는 유저 스택에서 넘겨받은 메모리에 불법적인 접근이 일어나는지 검사하고, 넘겨받은 file descriptor 값에 해당하는 파일이 있는지 검사한 다음, 해당 파일에서 읽고 있는 위치의 다음 byte를 반환한다.



유저 스택에서 넘겨받은 메모리에 불법적인 접근이 일어나는지 검사하고, 넘겨받은 file descriptor에 해당하는 파일이 존재하는 지 검사한다. 마지막으로 현재 읽고 있는 위치의 byte를 반환한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

유저 스택에서 넘겨받은 메모리에 불법적인 접근이 일어나는 지 검사한다. 다음으로, 파일을 읽기 전에 lock을 활성화한다. file descriptor에 해당하는 파일이 존재하면, 그 파일을 읽은 뒤, lock을 해제한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

유저 스택에서 넘겨받은 메모리에 불법적인 접근이 일어나는 지 검사한다. 다음으로 lock을 건 뒤, 해당 파일에 쓰기 작업을 수행한다. 수행 작업이 끝나면 lock을 해제한다.

해당 시스템 콜에서 file\_write함수를 호출하기 전 file\_deny\_write를 호출하고, 함수를 호출한 뒤 file\_allow\_write를 호출하여 다른 프로세스가 접근하는 것을 막고자 했으나, 위처럼 구현할 경우 여러 테스트케이스들이 통과되지 못하여 적용할 수 없었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

sysexit 함수에 file descriptor array에 해당하는 파일을 모두 닫는 코드를 추가하였다.

5. process.c

텍스트이(가) 표시된 사진

자동 생성된 설명

process\_execute 함수에 동기화를 위한 코드를 추가하였다. 세마포어 L을 sema\_down하고, child 리스트를 탐색하며 err\_flag가 1인 쓰레드를 찾으면 process\_wait을 호출하여 parent가 child보다 먼저 종료되는 것을 방지하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

start\_process 함수에서는 현재 실행중인 쓰레드의 부모 쓰레드의 세마포어 L을 sema\_up해주었다. 다음으로, load함수가 false값을 반환했을 경우에 err\_flag값을 1로 바꾸고, sysexit(-1)을 호출하였다. 처음에 해당 코드를 추가하지 않아 multi-oom 테스트케이스를 통과하지 못하였는데, 해당 코드를 추가하여 해결하였다.

6. exception.c

텍스트이(가) 표시된 사진

자동 생성된 설명

page\_fault 함수에 not\_present 값이 true일 경우, 즉 writing r/o page가 아닌 not-present page일 경우 sysexit(-1)을 호출하였다.

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

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

make check시에 동기화와 관련된 테스트케이스와 읽기-쓰기와 관련된 테스트케이스를 통과하지 못하였다. 오류 메시지를 참조하면 불리지 않아야 할 곳에서 sysexit(-1)이 호출되거나 세마포어가 잘못 사용된 것으로 보이지만, 디버깅을 해도 잘못 호출된 위치를 찾을 수 없었다. 또한, file\_deny\_write 함수를 적절한 곳에 사용하지 못한 것도 모든 테스트케이스를 통과하지 못한 원인이라고 보여진다.