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

담당 교수 / 분반 :박성용교수님 / 1반

이름 / 학번 :김은원 / 20192134

개발 기간 : 2022.10.04 ~ 2022.10.25

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

Create, remove, open, close, filesize, read, write, seek, tell 등 system call을 추가 구현한다.

파일 접근할 때 필요한 file descriptor를 구현한다. File synchronization을 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 파일에 접근할 때 사용하는 개념이다. 0이 아닌 정수 값을 가지며 프로세스가 파일을 오픈할 때 커널은 해당 프로세스가 사용하지 않는 값 중 작은 값을 할당해준다. 프로세스가 열려 있는 파일에 접근할 때는 system call을 이용해 file descriptor 값을 통해 파일에 접근한다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

System call handler를 통해서 명령어 Create, remove, open, close, filesize, read, write, seek, tell 등을 실행하여 명령어에 맞는 결과를 나타내야 한다.

3. Synchronization in Filesystem

각 프로세스는 공유데이터에 접근하는 중요한 섹션이 있다. 프로세스가 한 섹션에서 실행될 때 다른 프로세스가 그 섹션에서 실행되지 않도록 해야 한다. 이를 해결하는 것이 바로 동기화이다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

File descriptor를 thread 내부 배열로 구현하였다.

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: 입력 받은 파일이 NULL이면 종료, 아니면 입력 받은 크기 만큼 파일을 생성한다.

Remove: file이 NULL이면 종료, 아니면 파일을 제거한다.

Open:파일이 NULL이면 종료하고 아닌 경우 File descriptor 중에서 비어 있는 배열을 찾아 정수를 return 한다.

Filesize: 입력 받은 정수로 현재 thread의 file descriptor를 확인해 NULL인 경우 종료하고 아니면 해당 파일 길이를 리턴 한다.

Read: 입력 받은 file descriptor로 현재 파일이 NULL이 아니면 문자열이 끝날 때까지 읽어 인덱스 값을 반환한다.

Write: 출력하는 함수이고 fd가 3이상일 경우 버퍼에 있는 것을 읽어 출력한다.

Seek: 입력 받은 fd, position으로 파일이 NULL일 경우 종료하고 아니면 파일 위치를 이동한다.

Tell: 입력 받은 fd로 파일이 NULL일 경우 종료하고 아니면 파일 위치를 알려준다.

Close: 파일이 NULL인 경우 종료하고 아니면 NULL 값을 넣고 파일을 종료한다.

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

동기화란 여러 프로세스가 동시에 실행되어도 공유 데이터의 일관성을 보장하는 것이다. 공유 데이터의 일관성을 보장하기 위해서는 하나의 프로세스 또는 스레드만 실행하여야 된다. 이를 mutual exclusion이라 하는데 이를 보장하기 위해서는 lock을 사용해야 한다. Critical section에 들어가기 전에 acquire lock을 걸어준다. 만약 락을 얻는 데 성공한다면 critical section을 실행하고 실패했다면 다른 프로세스가 락을 반환할 때까지 기다려야 한다. Semaphore도 critical section에 접근하도록 하는 장치이다. 프로세스의 순서를 정할 수 있다. 이를 통해 동기화를 구현할 수 있다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

File descriptor : 10/5

System call:10/5~10/25

Synchronization: 10/25

* 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**:

Threads/thread.h 에 fd를 선언한다. 파일 포인터 fd는 배열로 구현한다.

2. **System Calls**: userprog/syscall.c 에 handler를 구현하고 create, close 등의 함수를 구현한다. syscall.h에 구현할 함수 (create, close 등)을 선언한다.

3. **Synchronization** in Filesystem: Lock, Semaphore를 어떻게 이용할 수 있는지 각각에 대해 설명:

동기화 작업을 해주어야 한다. Threads/thread.h에 semaphore load\_sema를 선언한다.

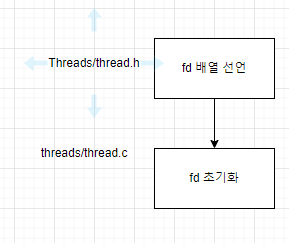
Threads/thread.c에서 sema를 초기화한다.

Lock은 open, read, write 함수에서 설정 해야한다. 시작 전에 lock\_acquire를 한 뒤 return 전에 lock\_release를 한다.

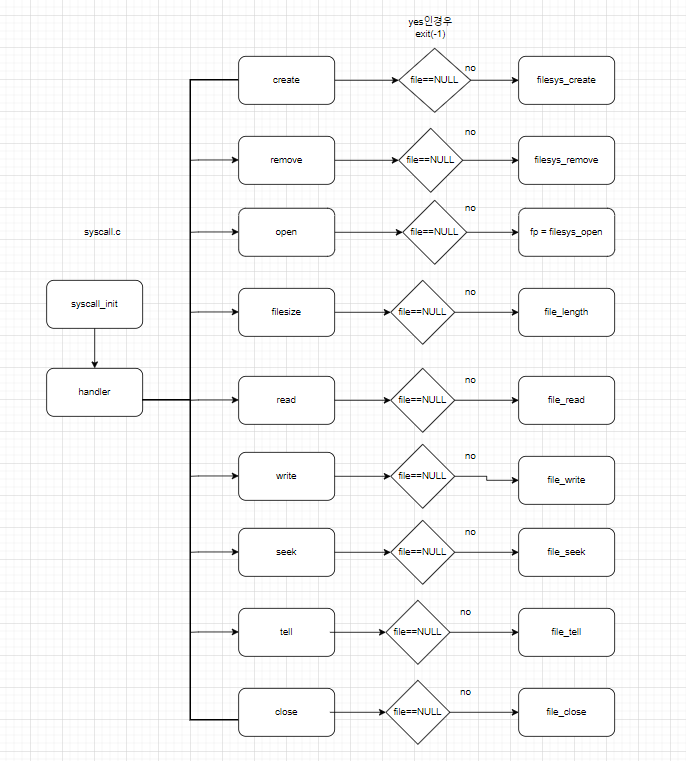
Process.c 소스코드 중 process\_execute과 start\_process함수를 수정하였다.

1. **연구 결과**
   1. **Flow Chart**

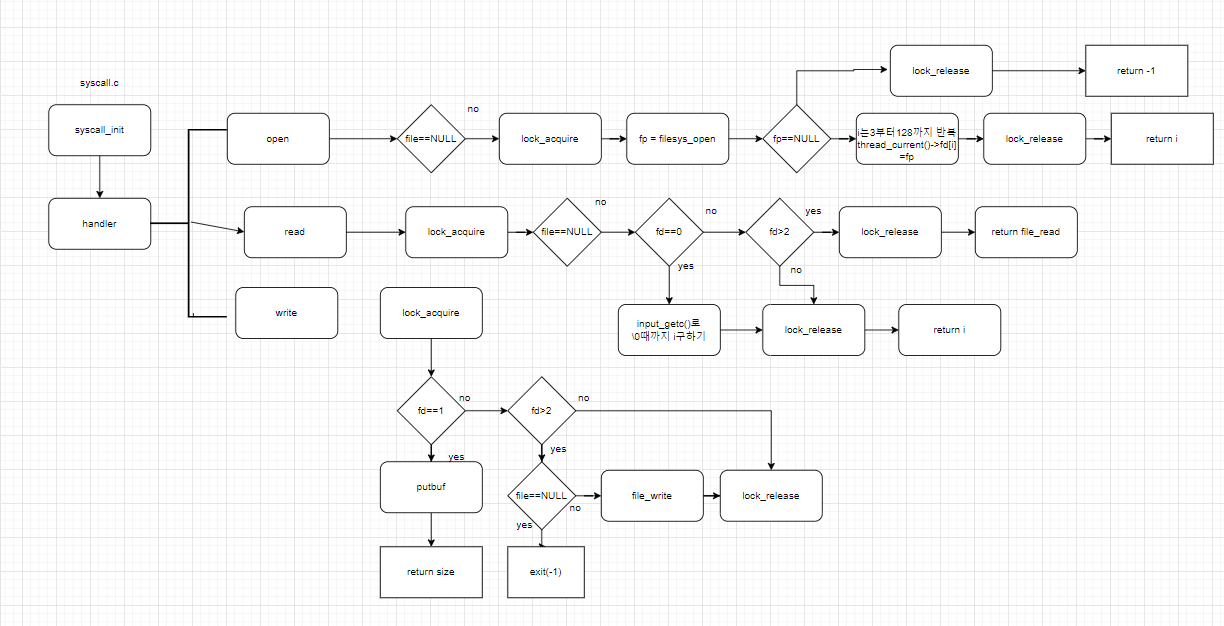
* II. B. 개발 내용의 각 3가지 항목에 대하여 Flow Chart 작성
  + 1. **File Descriptor**

****

* + 1. **System Calls**

****

* + 1. **Synchronization**

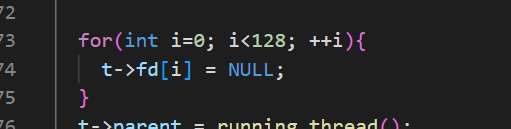


* 1. **제작 내용**
* II. B. 개발 내용의 각 3가지 항목에 대하여 실직적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
* 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명

1. File Descriptor: threads/thread.c에 fd를 선언하였다. 파일 포인터 배열로 선언하였다.



파일 포인터는 모두 NULL로 초기화 한다.



이는 thread.c에서 수행했다.

2. System Calls: proj1에서 수행한 system handler처럼 create 등의 함수를 구현했다.

**Create**: file이 NULL일 경우 종료하고 아니면 filesys\_create 함수를 호출한다. 이 함수는 inode를 생성하고 초기화한다. Free-map에서 indoe의 블록을 할당한다. File inode를 디스크에 기록하고 디렉터리 엔트리에 추가한다. 성공했다면 bit map에서 할당된 inode의 블록을 해제하고 root 디렉터리 inode 메모리를 해지하고 success를 반환한다.

**Remove**: file이 null이면 종료하고 아니면 filesys\_remove를 호출한다. Flesys\_remove는 root 디렉터리에 inode 메모리를 할당하고 root 디렉터리에서 검색한다. 만약에 파일이 존재하면 true 아니면 false를 반환한다. true이면, root디렉터리에서 메모리를 해지해 제거한다.

**Open**: 우선 파일이 bad pointer인지 확인하는 check\_user\_vaddr를 추가했다. 그리고 NULL인 경우 바로 종료한다. 파일 포인터를 filesys\_open을 통해 받는다. Filesys\_open은 file 자료구조를 생성하고 초기화해 파일 포인터를 반환하는 함수이다. 파일 포인터가 NULL이면 -1을 반환하고 아닌 경우 NULL인 파일 디스크립터를 찾는다. 만약 스레드 이름과 동일하면 file\_deny\_write을 통해 방지해야 한다. 그렇지 않다면 파일 디스크립터를 직접 할당한다.

**Filesize**: 현재 파일이 NULL인 경우 종료하고 아니면 file\_length를 호출한다. File\_length는 파일의 크기를 반환한다.

**Read**: 파일이 bad pointer인지 확인하는 check\_user-vaddr를 추가하였다. 파일이 NULL을 가리키는 경우 바로 종료한다. 파일 디스크립터를 이용해 파일 객체를 검색하고 fd(file descriptor)가 0일 경우 키보드 입력을 버퍼에 저장한다. 그 뒤 버퍼의 저장한 크기를 반환한다. 0이 아닐 경우는 file\_read를 호출하는데 이 함수는 파일의 데이터를 크기만큼 저장하고 읽은 바이트 수를 반환한다.

**Write**: 파일이 bad pointer인지 확이나는 check\_user\_vaddr를 추가했다. Fd가 1인경우 바로 버퍼에 출력하고 size를 반환한다. 파일 디스크립터를 이용해 파일이 NULL인지 확인하고 NULL인 경우는 바로 종료한다. 아닌 경우, file\_write 함수를 반환한다. File\_write는 filesys/file.c에 있는 함수이다. 버퍼에 저장된 데이터를 크기만큼 파일에 기록하고 바이트 수를 리턴한다.

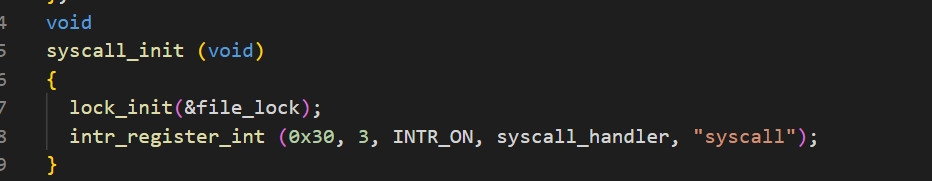
**Seek**: 열린 파일의 위치로 이동한다. Position은 현재 위치를 기준으로 이동할 거리를 뜻한다. File descriptor를 통해 NULL이면 종료하고 아닌 경우 file\_seek 함수를 이용한다. File\_seek은 파일 위치를 이동하는 함수이다.

**Tell**: 파일 위치를 알려주는 함수이다. File descriptor를 이용해 NULL일 경우 종료하고 아니면 file\_tell함수를 이용한다. File\_tell 함수는 file의 offset를 알려준다. 열린 파일의 위치를 반환한다.

**Close**: file descriptor를 이용해 NULL일 경우 종료하고 아니면 현재 파일을 NULL로 바꾸고 file\_close함수를 이용해 종료한다. File\_close는 Fd에 해당하는 entry를 초기화한다.

3. Synchronization in Filesystem: Lock, Semaphore를 어떻게 이용할 수 있는지 각각에 대해 설명:

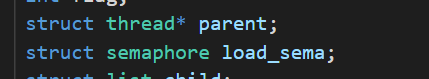
file에 대해 동시 접근이 일어날 수 있다. 따라서 lock과 semaphore를 사용한다. Pintos/src/threads/synch.h에 있는 lock을 사용한다. Struct lock filesys\_lock을 threads/thread.h에 추가했다. Read, write, open 시스템 콜에서 파일에 접근하기 전에 lock을 획득하도록 구현하고 파일 접근이 끝나면 lock release를 통해 lock을 해제했다. 또한 syscall\_init()함수에서 filesys\_lock 초기화 코드를 추가하였다. 다음은 Useprog/syscall.c에 있는 syscall\_init()함수에서 lock\_init(&file\_lock)을 추가한 코드이다.



Syn-read, syn-write를 통과하기 위해서는 동기화를 구현해야 한다. Open, write, read 함수에서 lock을 구현해야 한다. Lock\_init은 lock을 초기화하는 함수이다. Lock\_acquire는 다른 프로세스가 접근하지 못하게 잠그는 함수이다. Lock\_release는 lock을 해제하여 다른 프로세스가 접근할 수 있도록 한다.

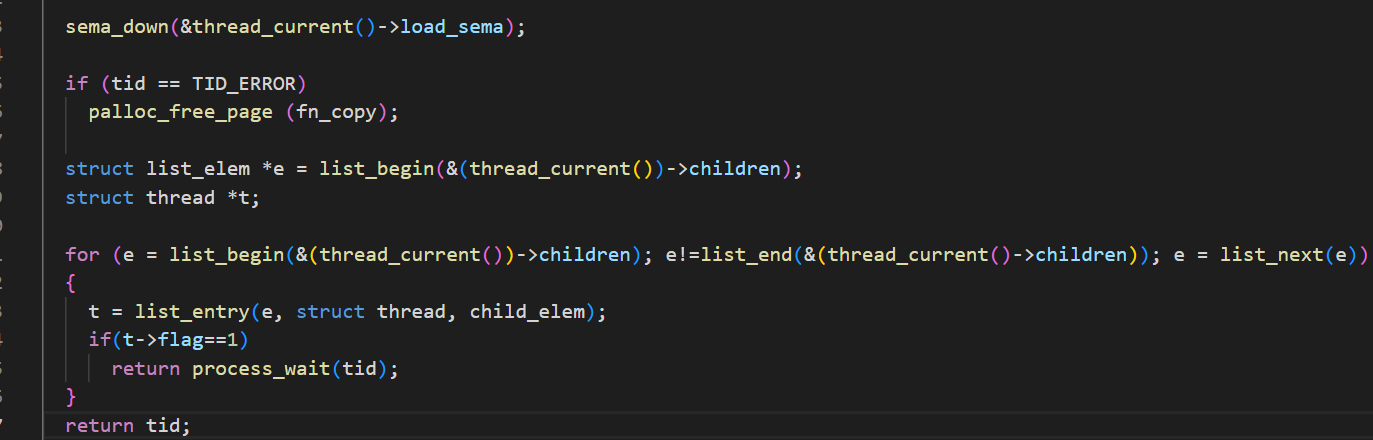
Read, write, open함수에서 file이 NULL인지 확인 후 lock을 잠근다. Return 전에 release를 하여 lock을 해제한다.

메모리 누수를 막기 위해서 process.c 파일을 수정한다. 이때 새로운 semaphore가 필요하여 threads/thread.h에 새로운 load\_sema를 추가한다.

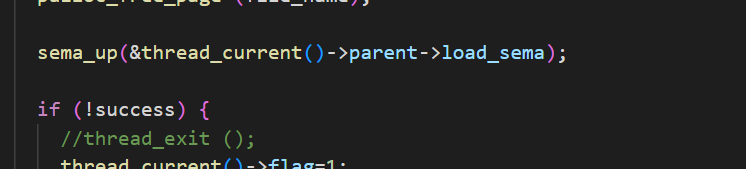
****

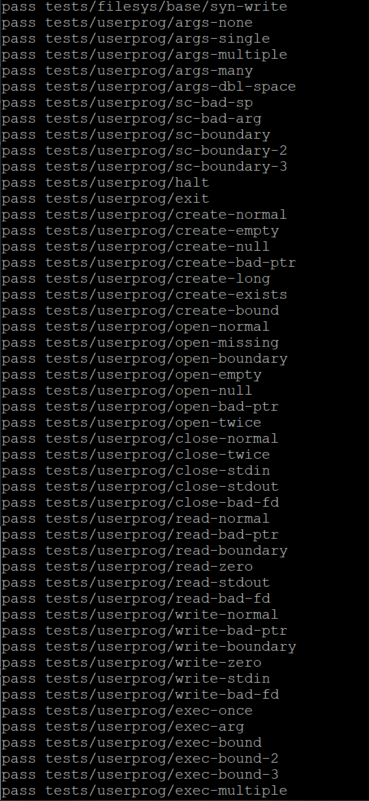
동일하게 semaphore도 초기화한다. Threads/thread.c에서 sema\_init를 한다.

Process.c에서 process\_execute 함수에서 thread\_create 다음 sema\_down을 한다. 이를 통해 list에 있는 것이 순차적으로 접근할 수 있게 한다. Process\_wait를 통해 child list 중 종료된 것이 있으면 회수해야 한다.



Start\_process에서 sema\_up을 하여 lock을 반환한다.

****

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