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

담당 교수 / 분반 : 김영재 교수님 / 2분반

이름 / 학번 : 한석기 / 20211606

개발 기간 : 2023/10/09 ~ 2023/10/17

1. **개발 목표**

Pintos의 Base File System을 이용해 create, remove, open, close, filesize, seek, tell의 system calls를 구현한다. file 관련하여 여러 일을 처리하는 system calls이며 semaphore와 lock을 적절히 사용하여 각 작업을 서로 침범하지 않게 하도록 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

1. File Descriptor

System call을 통한 file open 및 read, write는 file descriptor가 필요하다. file descriptor를 구현 및 관리함으로써 모든 thread가 파일에 접근할 수 있다. stdin은 0, stdout은 1, stderr는 2임을 주의하며 구현해야 한다.

2. System Calls

create, remove, open, close, filesize, seek, tell의 system calls를 구현해 파일 생성, 파일 삭제, 파일 열기, 파일 닫기, 파일의 크기, 읽고 쓰기 위치 변경, 읽고 쓰기 위치 반환의 기능을 하게 한다. 그리고 read와 write를 수정해 읽고 쓰기를 하게 한다.

3. Synchronization in Filesystem

다중 process에서 어느 파일에 접근할 때, 발생할 수 있는 문제를 semaphore, lock과 같은 struct를 활용해 관리한다.

* 1. **개발 내용**

1. File Descriptor

thread 구조체 내의 struct file 배열로 선언할 것이다. 배열은 접근이 빠르고 관리가 용이하기 때문에 적절하다.

2. System Calls

create: 주어진 이름을 가진 새로운 파일을 주어진 사이즈에 맞게 생성한다. 성공하면 true를 return하고 실패하면 false를 return한다.

remove: 주어진 이름을 가진 파일을 삭제한다. 성공하면 true를 return하고 실패하면 false를 return한다.

open: 주어진 이름을 가진 파일을 오픈한다. 오픈이 성공하면 file descriptor를 return하고 실패하면 -1을 return한다.

close: file descriptor에 맞는 파일을 닫는다.

filesize: file descriptor에 맞는 파일의 size를 return한다.

seek: file descriptor에 맞는 파일의 다음 읽거나 쓸 위치를 주어진 위치로 바꾼다.

tell: file descriptor에 맞는 파일의 다음 읽거나 쓸 위치를 return한다.

read: file descriptor에 맞는 파일을 size만큼 읽는다. 실제로 읽은 bytes 수를 return하고 file을 읽을 수 없다면 -1을 return한다. file descriptor가 0이라면 stdin이므로 input\_getc()를 이용해 입력을 읽는다.

write: file descriptor에 맞는 파일을 size만큼 쓴다. 실제로 쓴 bytes 수를 return하고 size보다 작을 수 있다. file descriptor가 1이라면 stdout이므로 console에 putbuf()를 통해 쓴다.

3. Synchronization in Filesystem

semaphore를 이용해 parent thread와 child thread 사이의 기다림을 생성한다. parent thread는 종료할 때, 모든 child thread가 종료된 후에 종료할 수 있으며 이것을 semaphore를 통해 구현한다. 그리고 lock을 이용해 file read, write 충돌을 관리한다. 각각의 과정과 처음과 끝에 lock을 걸고 풀고 과정을 진행한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

2023.10.09 ~ 2023.10.10: 구현해야 할 system calls manual 읽기

2023.10.11: Synchronization 방법 생각

2023.10.12 ~ 2023.10.15: system calls 구현 및 Synchronization

2023.10.16: make check 및 오류 해결

2023.10.17: 보고서 작성

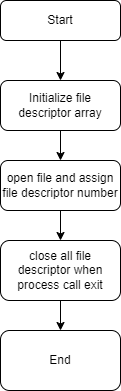
* 1. **개발 방법**

file descriptor를 thread 구조체 내에 struct file 배열로 선언하고 wait의 상태를 나타내는 wait status와 parent를 저장할 포인터, 그리고 parent process와 child process 사이 기다림을 주기 위한 semaphore를 추가한다. file read, write 충돌을 막기 위한 lock 구조체는 syscall.c에 추가해준다.

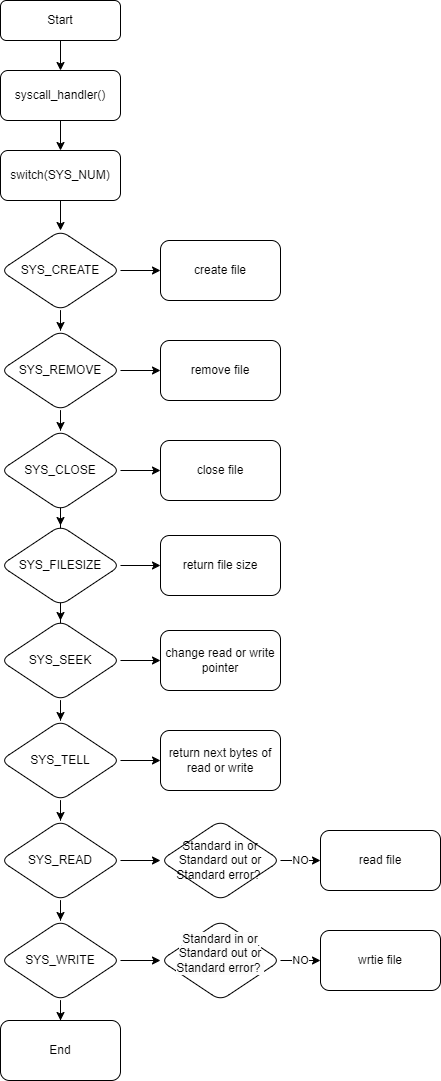
create, remove, open, close, filesize, seek, tell의 함수를 추가해준다. 각각의 기능은 위에 설명되어 있다. read, write, exit 함수를 수정해준다. read, write는 file descriptor와 lock, base file system을 활용해 구현한다. exit에서는 file descriptor 관리하는 코드를 추가한다. 그리고 process.c에서 parent process와 child process 사이 기다림을 주기 위한 semaphore로 관리를 해주고 wait status에 따른 관리도 추가해준다.

exception.c에서 file read, write 관련하여 예외처리 되지 않은 부분을 확인하고 예외처리 해준다.

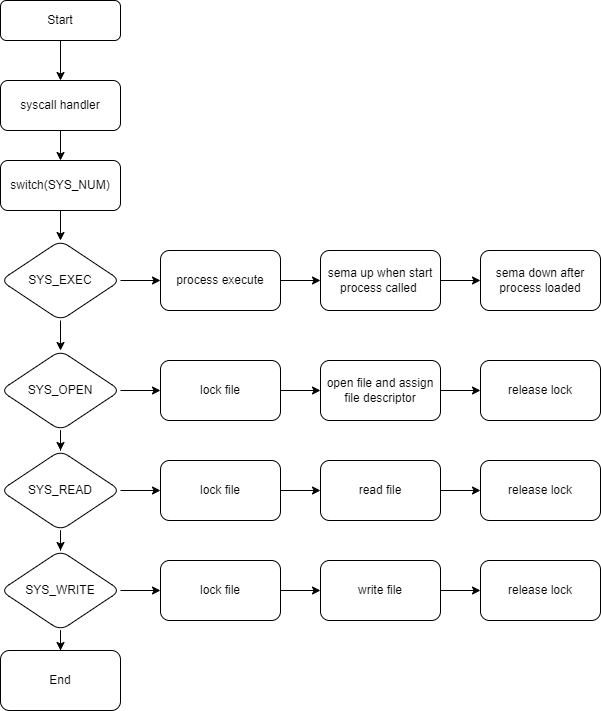
1. **연구 결과**
   1. **Flow Chart**
      1. file descriptor



* + 1. System Calls



* + 1. Synchronization in Filesystem

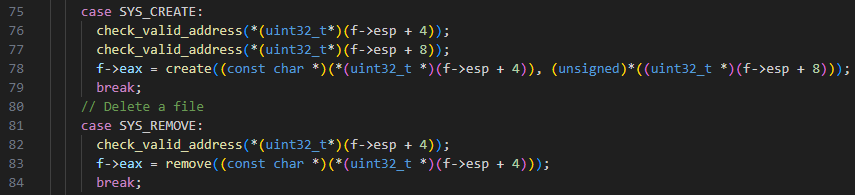


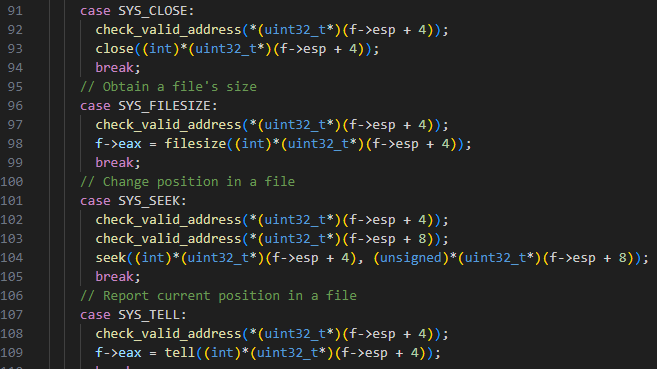
* 1. **제작 내용**

실제 구현 순서대로 따라가면 다음과 같다.

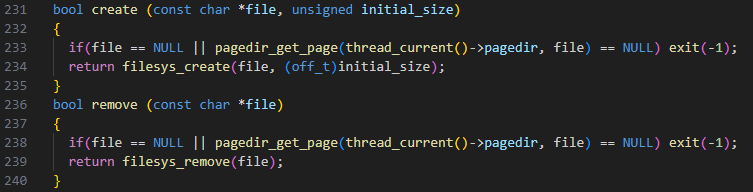
먼저, Base File System APIS만을 이용해 구현할 수 있는 System calls를 구현했다.

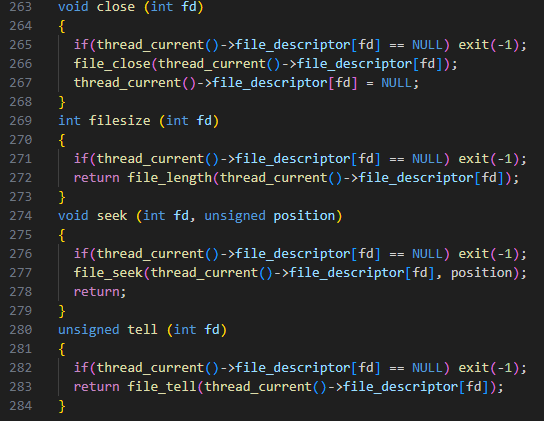
파일은 userprog/syscall.c이다.



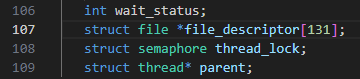


위는 각각 create, remove, close, filesize, seek, tell에 대한 system call의 switch문에서의 함수 호출이다. Case에 따라 arguments의 유효성을 확인하고 맞는 함수를 출력한다. 이것은 Project1에서 했던 과정과 동일하다.

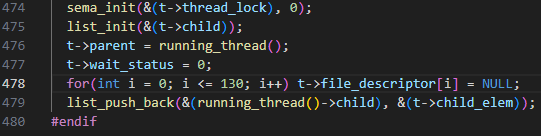




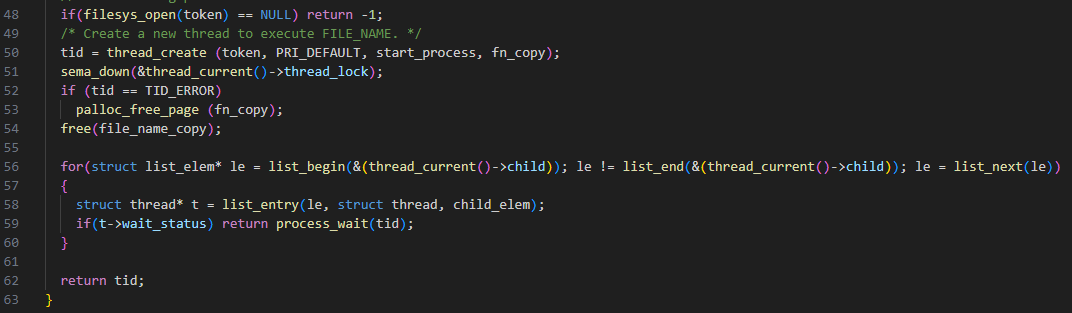
System calls API를 이용한 함수 구현이다. create함수부터 tell함수까지 인자에서 나올 수 있는 예외를 처리해주고 각각의 System call에 맞는 API들을 이용한다. 이 API들은 Pintos에서 제공하는 Base File System을 이용하게 해준다.



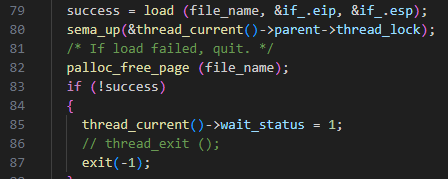
위는 threads/thread.h에 thread 구조체 내 #ifdef USERPROG ~ #endif 사이에 추가한 요소들이다. wait\_status는 load가 성공하지 않을 시, parent에게 알리는 용도로 사용한다. file\_descriptor는 file descriptor를 관리하는 배열이다. thread\_lock은 parent process와 child process 간의 기다림을 위한 semaphore이고 parent는 child의 parent다.



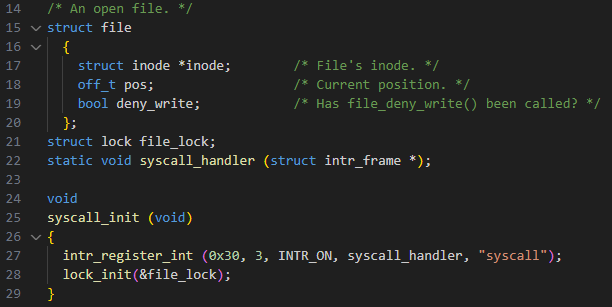
threads/thread.c의 init\_thread 부분이다. thread\_lock을 초기화 시키고 부모 쓰레드를 자식 쓰레드의 parent에 넣어주고 wait\_status도 초기화 시킨다. 그리고 file descriptor 배열을 초기화한다.



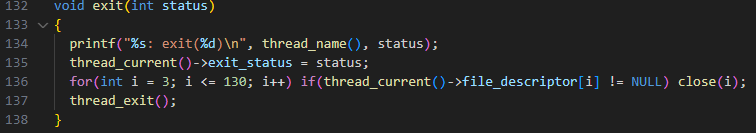
userprog/process.c의 process\_execute 함수 중 sema\_down으로 lock을 풀어주는 곳이다. 그리고 마지막에 child threads를 돌며 child가 종료될 때까지 기다리는 코드를 추가해준다. wait\_status를 exit\_status로 관리하는 우를 범하다가 test에서 wait 관련된 test가 잘 이뤄지지 않는 것을 확인했다.



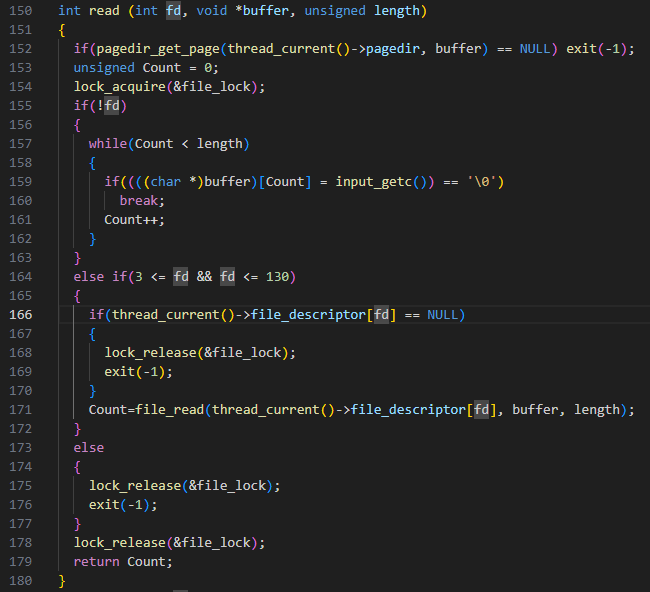
userprog/process.c의 start\_process 함수 중 sema\_up으로 lock을 걸어주는 곳이다. child thread기 때문에 parent로 접근해 thread\_lock을 걸어준다. 이것 때문에 thread 구조체에 parent를 추가한 것이다. 그리고 load가 성공적으로 이뤄지지 않을 시, wait status를 1로 주고 비정상 종료한다.



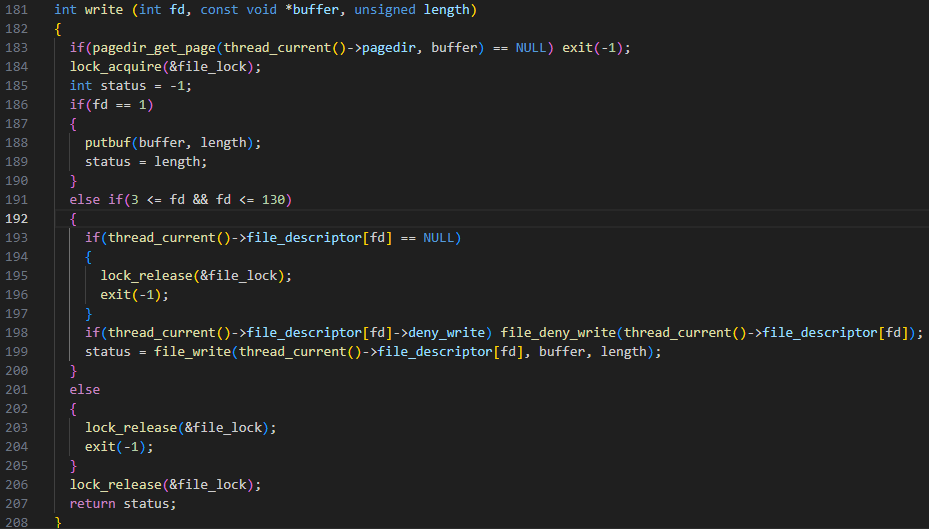
userprog/syscall.c의 file 구조체 선언과 file 읽기 쓰기를 관리할 lock 구조체, 그리고 lock 구조체를 초기화 하는 코드다. file 구조체는 filesys/file.c의 file 구조체를 그대로 가져왔다. deny\_write는 open함수와 write 함수에서 관리한다 deny\_write 관련 함수를 따라가면 filesys/inode.c의 inode\_write\_at에 deny\_write\_cnt가 0이 아니면 종료하는 코드가 있다. file\_deny\_write()가 call 되면 deny\_write\_cnt가 up되고 filesys/file.c의 file\_close함수의 file\_allow\_write 호출을 통해 deny\_write\_cnt가 down되는 것을 확인할 수 있다.



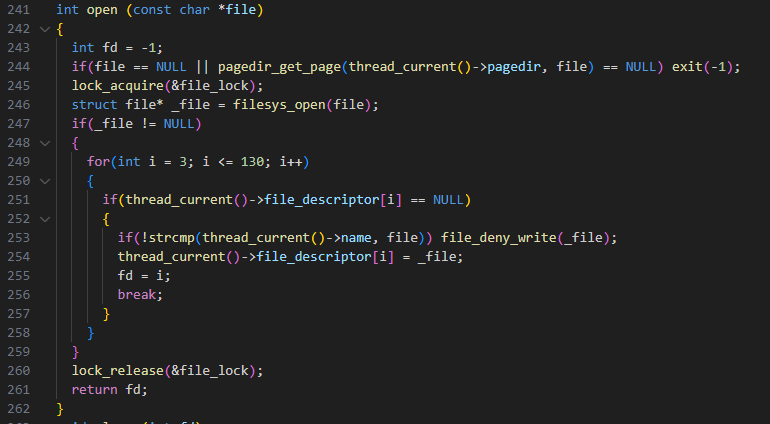
userprog/syscall.c의 exit 함수 수정사항이다. file\_descriptor 배열을 추가해줬으므로 프로세스를 종료할 때, close되지 않은 파일을 모두 확인하고 close한다.



userprog/syscall.c의 read함수 수정사항이다. 먼저 read과정 시작이므로 file lock을 걸어준다. fd가 stdin(0)이면 Project1에서 구현했던 코드를 실행하고 3이상 130이하(최대 128개의 파일을 열수 있다 했으므로)이면 file\_read 함수를 이용해 read한다. 그 과정에서 발생할 수 있는 예외처리를 해주고 마지막에 lock을 푼다.



write도 비슷한 과정을 따른다, lock을 걸고 fd가 stdout(1)이면 Project1에서 구현했던 코드를 실행한다. fd가 3이상 130이하이면 file\_write 함수를 이용해 write한다. 이 때, deny\_write를 확인한다. 그리고 마지막에 lock을 푼다.

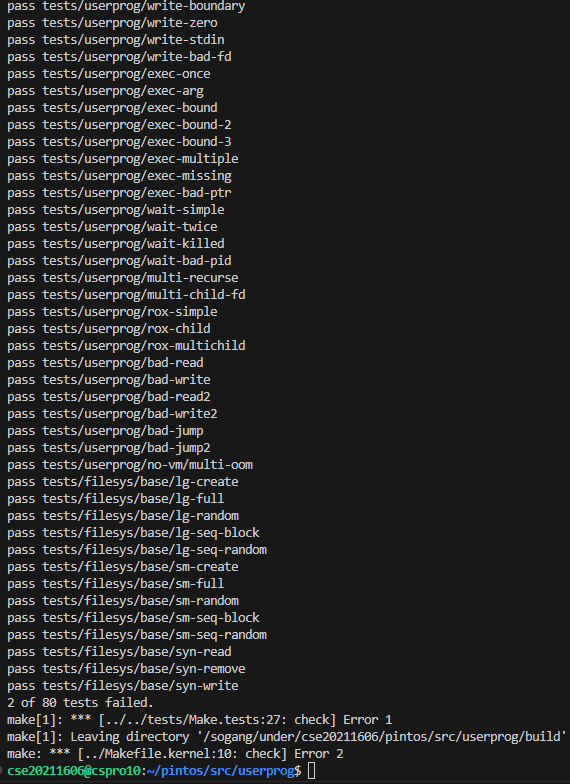


open은 lock을 걸고 filesys\_open을 통해 file을 open한다. 오픈한 파일이 NULL이 아니라면 3번부터 130번까지의 file descriptor를 돌며 빈 곳을 찾는다. 빈 곳을 찾고 현재 thread의 이름과 파일 이름이 같은지 확인하고 file\_deny\_write를 호출한다. 해당 fd 번호에 \_file을 저장하고 lock을 해제한 후 fd를 return한다.

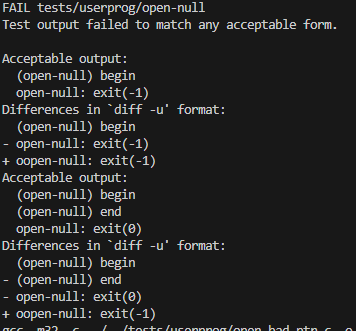


userprog/exception.c의 page\_fault함수이다. not\_present는 할당되지 않은 read/write할 때, 할당된 메모리영역인지 확인하는 것인데 test 시, 위의 예외처리를 하지 않는다면 bad-read, bad-write, bad-jump, multi-oom의 test가 FAIL한다.

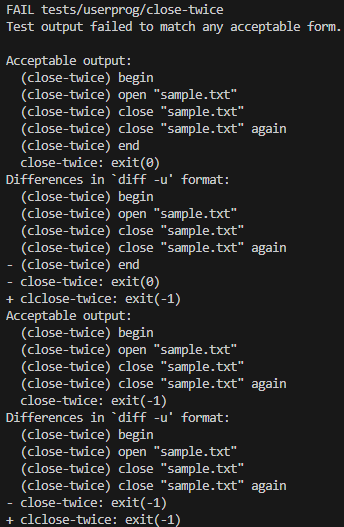
* 1. **시험 및 평가 내용**



make check를 1회 try했는데 2개가 실패했다.

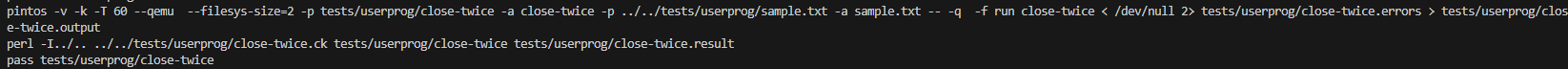


서버 문제인지 oopen이런식으로 o가 두 번 들어가서 비정상 종료인 것을 확인했다.



두번째 FAIL도 마찬가지였다. clclose 이런 식으로 문제가 발생했다.





서버 문제가 맞았다. 그대로 make clean 후 두번째 try에서는 두 오류 모두 PASS한 것을 확인할 수 있었다.