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

담당 교수 / 분반 : 김영재 / 2분반

이름 / 학번 : 이용욱 / 20191626

개발 기간 :2023.10.20 ~ 2023.10.28

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

해당 프로젝트에선 프로젝트1에서 구현하지 않은 system call과 signal handler들을 구현하는것을 목표로 한다. 해당 시스템 콜들은 커널이 파일을 처리하는 시스템 콜들로, SYS\_CREATE, SYS\_REMOVE, SYS\_OPEN, SYS\_CLOSE, SYS\_FILESIZE, SYS\_SEEK, SYS\_TELL을 새롭게 추가 구현하고, SYS\_WRITE과 SYS\_READ를 STDIN과 STDOUT외의 파일 입출력에 관해서도 기능하도록 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

2. (이번 프로젝트에서 구현해야 하는) System Calls

3. Synchronization in Filesystem

file descriptor을 통해 커널이 타겟 파일들의 주소를 저장한다. system call을 통해 특정 file들에 접근하고 이용하기 위해서는 file descriptor가 필요하다.

file descriptor를 구현하여 각각의 thread가 파일에 접근하도록 만든다.

create, remove, open, close, filesize, seek, tell을 각각의 목적에 맞게 구현하고 프로젝트 1에서 구현한 read와 write를 표준 입출력이 아닌 file또한 접근할 수 있도록 수정한다.

여러 process에서 하나의 파일에 접근하고 사용할 때 일어날 수 있는 동기화 문제를 해결한다. 커널이 제한할 수 있도록 세마포어, 락과 같은 방식을 사용하여 동시 접근을 해결한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

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)

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

File descriptor를 구현하기위해 사용한 thread 구조체에 정적 배열을 추가한다. 그리고 thread가 생성될 때 이 구조체 배열을 NULL로 초기화시키고 이 정적 배열의 int fd[128]을 사용하여 파일들에 접근할 수 있게 된다. 이 구조체 배열에 file descriptor를 저장하고, 다른 동작을 실행할 때 이를 사용하여 원하는 파일에 접근하도록 한다.

ㄴ 아래는 구현할 각 system call에 대한 설명이다.

1. SYS\_CREATE (bool create (const char \*file, unsigned initial\_size))

파일명과 초기 size 를 입력받아 해당 파일명으로 파일을 생성한다. 생성 성공이나 실패 여부에 따라 true, false를 반환한다. 이 때 파일을 open하지 않는다.

1. remove (bool remove (const char \*file))

파일명을 입력 받아 remove함수를 통해 해당 파일명의 파일을 제거한다. 성공이나 실패 여부에 따라 true, false를 반환한다.

1. open (int open (const char \*file))

파일명을 입력받아서 해당 파일명의 파일을 연다. 그 file의 index를 반환한다.

1. close (void close (int fd))

입력받은 fd로 해당 파일을 닫는다. 이 때 해당 file descriptor는 NULL이 된다.

1. filesize (int filesize (int fd))

해당 'fd'의 파일디스크립터를 입력받아서 해당 파일의 사이즈를 byte 단위로 반환한다.

1. read (int read (int fd, void \*buffer, unsigned size))

기존에 구현했던 표준입력 (fd == 0) 외에 다른 파일의 read에 대해서도 구현한다. 'fd'의 file descriptor를 갖는 해당 파일에 'buffer'의 'size'만큼 읽는다. 이 때 표준 입력인 경우 input\_getc() , 파일 입력인 경우 file\_read를 사용한다.

1. write (int write (fd, const void \*buffer, unsigned size))

기존에 구현했던 표준출력 (fd == 1) 외에도 다른 파일의 write에 대해서도 구현한다. 'fd'의 file descriptor를 갖는 해당 파일에 'size'만큼 buffer 로부터 쓴다. 이 때 표준 출력인 경우 putbuf() , 파일 입력인 경우 file\_write를 사용한다.

1. seek (void seek (int fd, unsigned position))

fd를 입력받아서 해당 파일 커서를 입력받은 위치로 이동한다

1. tell (unsigned tell (int fd))

'fd'를 입력받아서 열린 파일에 쓰거나 읽을 다음 바이트의 위치를 반환한다.

이 때 syncronization을 위해서는 filesys\_lock변수를 선언하여 이용한다.

위의 system call handler들의 호출 직후에 lock을 설정 그리고 끝나기 직전에 lock을 해제 한다. 이 대 lock\_acquire과 lock release함수를 사용하여 여럿이 접근할 때 한번에 하나의 프로세스가 접근하도록 하는 것이다. 내부적으로는 semaphore를 사용하기에 큰 차이가 없다.

또한 process간의 synchronization에서는 semaphore를 사용하여 문제를 해결한다. process\_execute 에서 sema\_down을 호출하고, start\_process에서 sema\_up을 호출하여 synchronization문제를 해결한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

23.10.20 ~ 23.10.22: create, remove, open, close, filesize, seek, tell 등의 system call 구현

23.10.23 ~ 23.10.28: read, write 에 추가기능 구현 및 synchronization 문제 해결

22.10.28: 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

우선 file descriptor를 저장할 수 있는 자료구조를 pintos/src/threads/thread.h에 저장된thread 구조체에 추가한다.

그리고 thread.c의 init\_thread() 모든 fd를 NULL롤 초기화 하도록 한다.

추가로 exception.c에 프로세스의 정상적인 동작을 위해 not present에 관한 if문을 추가한다.

syscall.c에서는 추가로 구현해야하는 system call들을 구현한다. sdf

\create, remove , open, close, filesize, seek, tell함수를 구현하고, 기존의 read, write 함수를 수정한다. 그리고 syscall\_handler 함수에 위의 시스템 콜을 handle할 함수들 SYS\_CREATE, SYS\_REMOVE, SYS\_OPEN, SYS\_CLOSE, SYS\_FILESIZE, SYS\_SEEK, SYS\_TELL을 추가하여서 사용한다. 이 때 validation\_check로 invalid memory access를 처리한다. 또한 synchronization문제를 해결하기 위해 lock관련 함수들을 사용한다.

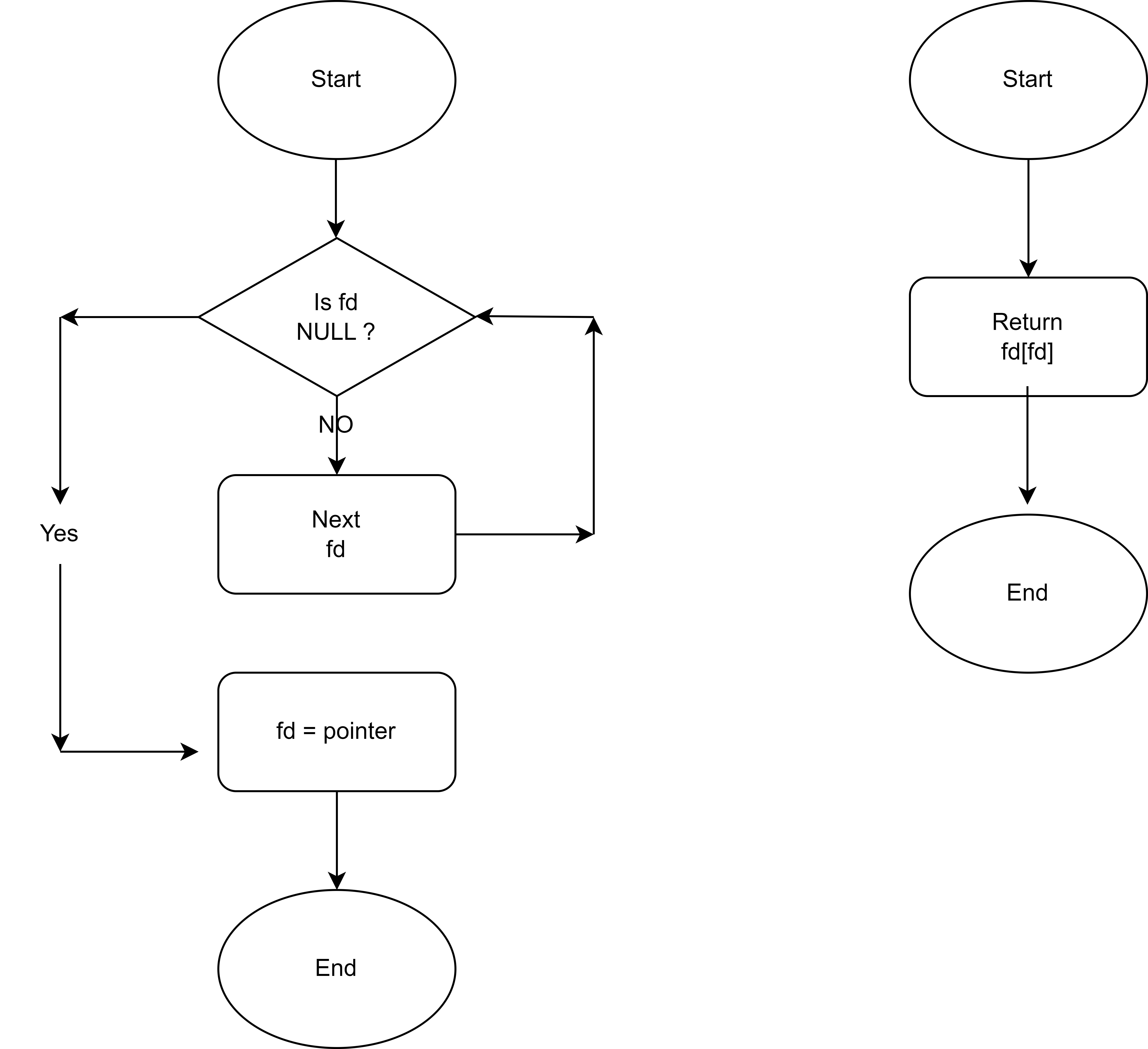
syscall.h에 구현한 system call 함수를 선언한다.

thread.h 에 동기화를 해결하기 위해 struct semaphore load\_child를 추가한다. 이를 process.c에서 process\_execute와 start\_process에서 사용한다.

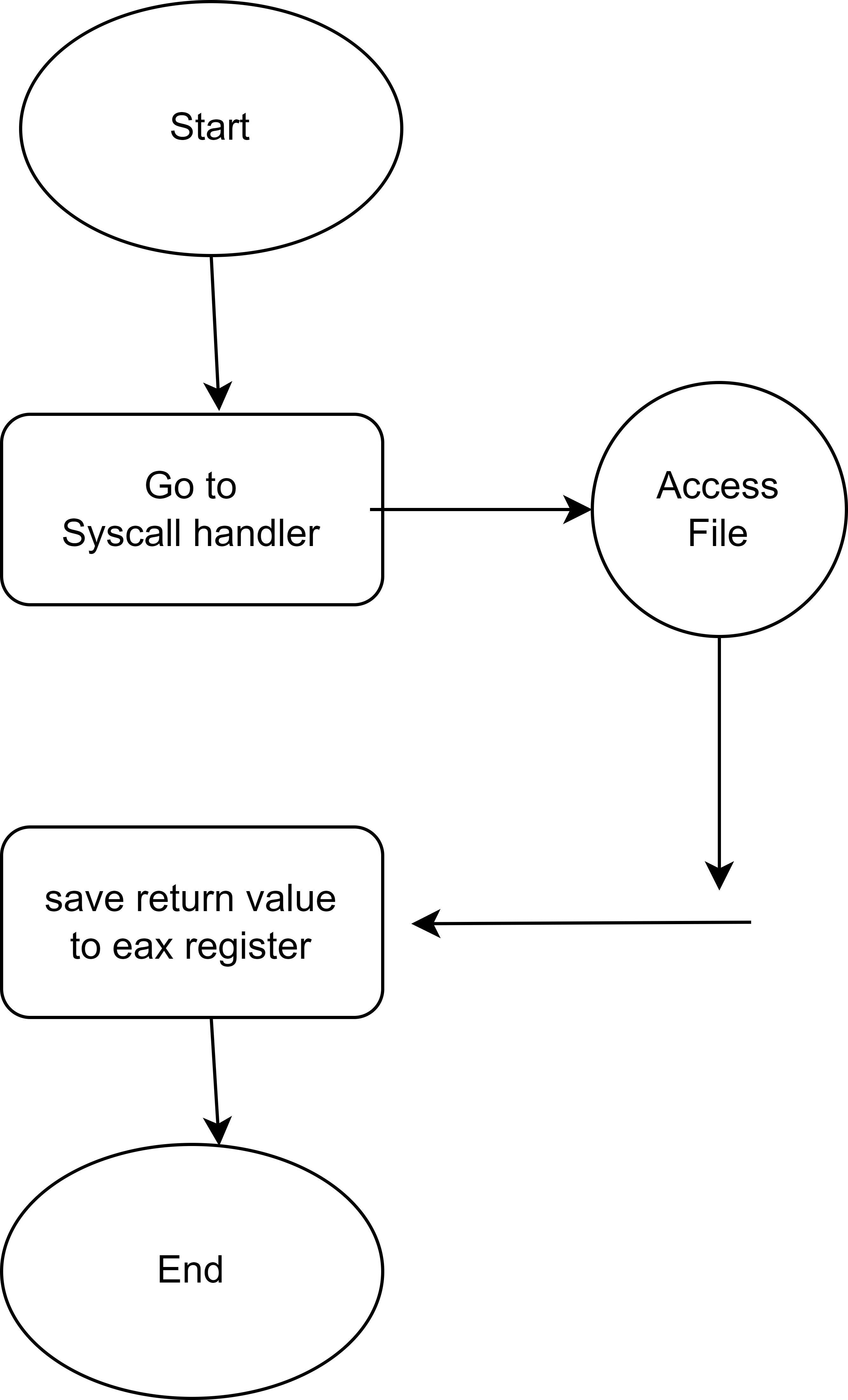
1. **연구 결과**
   1. **Flow Chart**

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

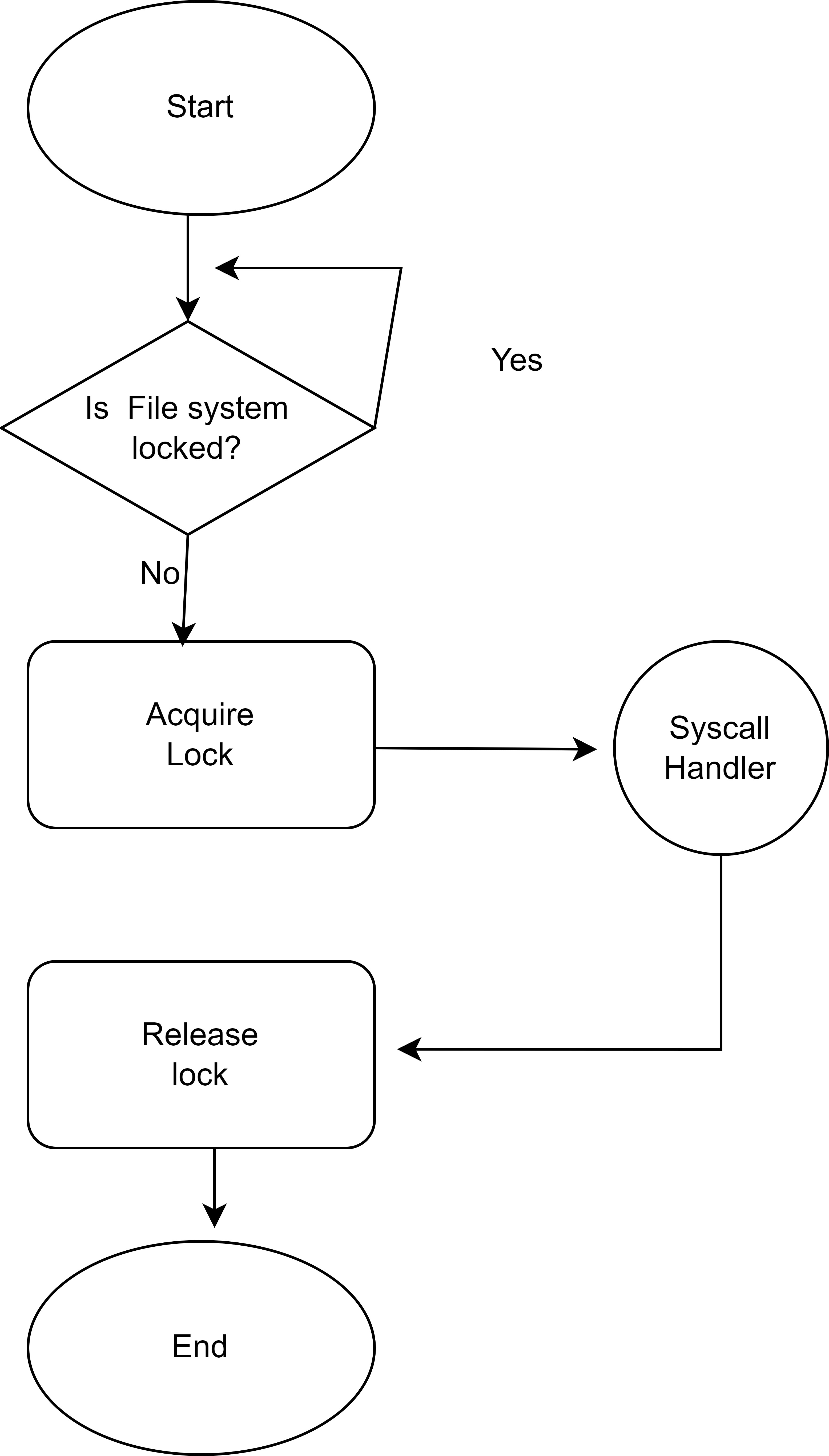
1) File descriptor 저장 및 접근



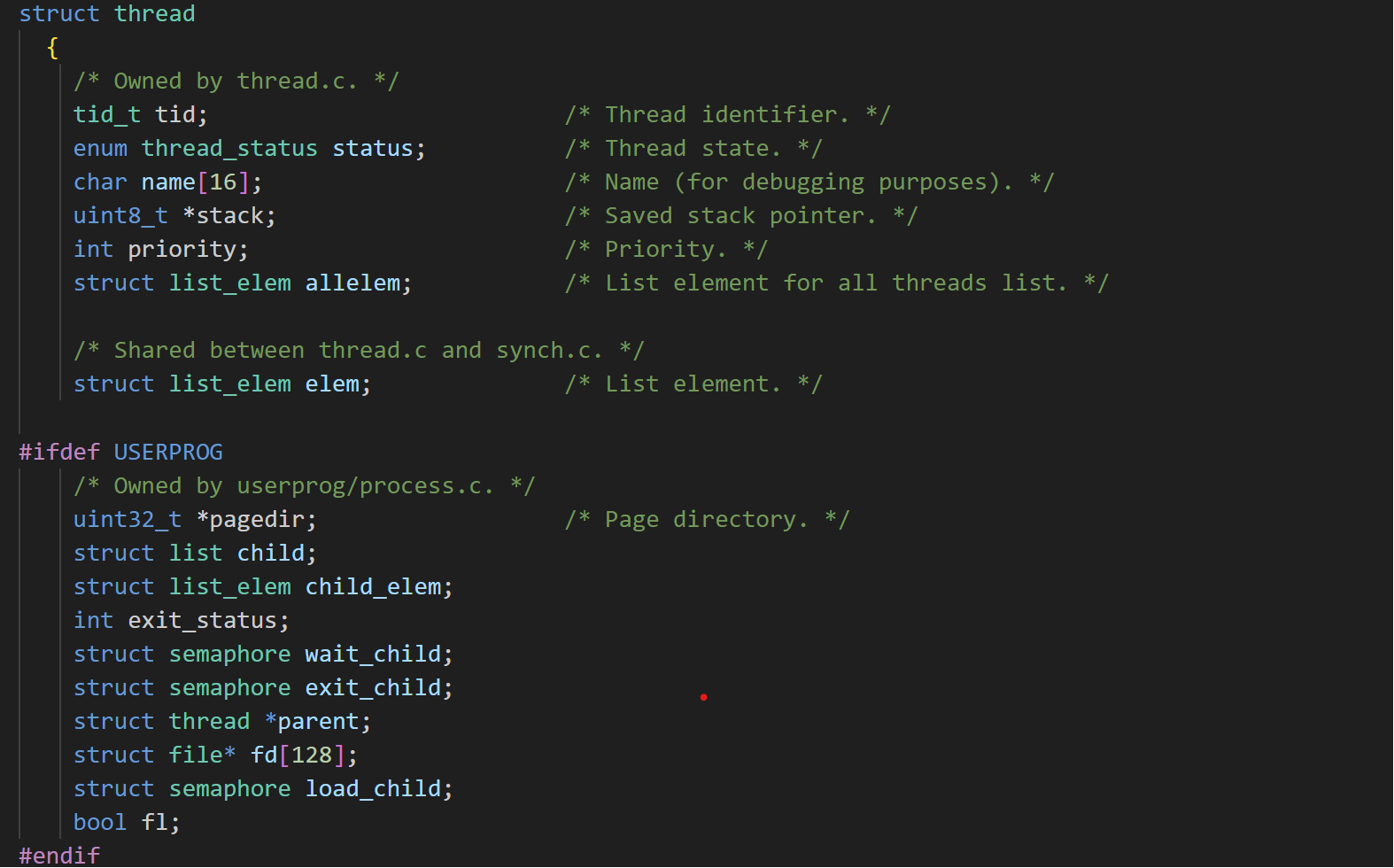
2) System calls



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) pintos/src/threads/thread.h 의 struct thread 구조체에서

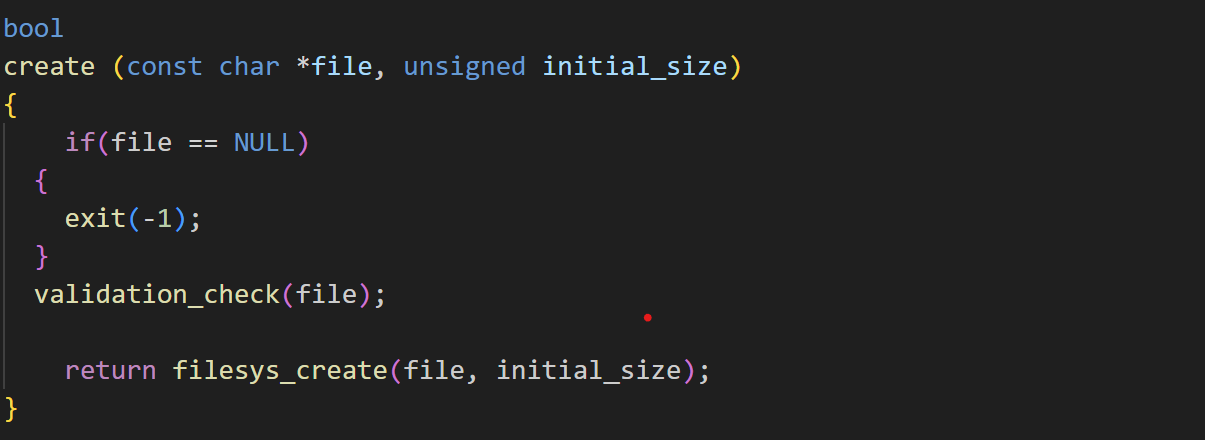
기존에 있던 내용에 file descriptor 사용을 위해서 struct file\* fd[128]을 추가한다.

그리고 pintos/src/threads/thread.c중 init\_thread함수에서



위와 같은 코드를 추가해서 thread가 생성될 때 모든 fd를 NULL로 초기화하고, semaphore 또한 초기화해준다.

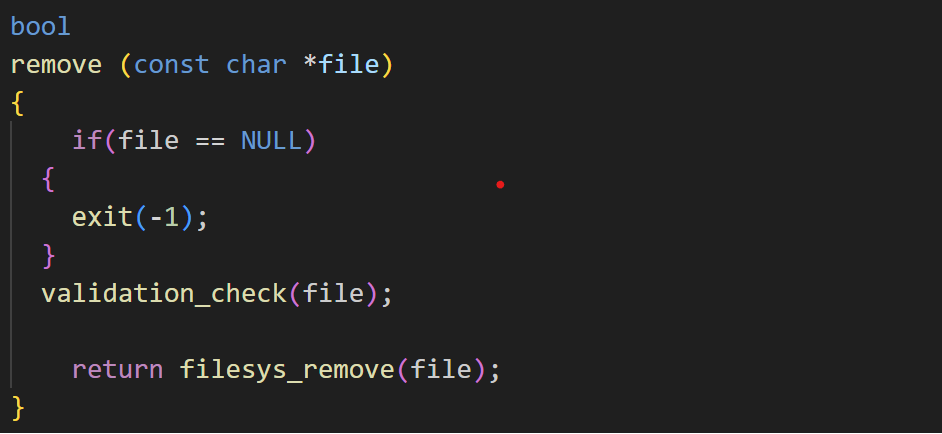
2) pintos/src/userprog/syscall.c에서 sytem call들을 추가 구현한다.



create

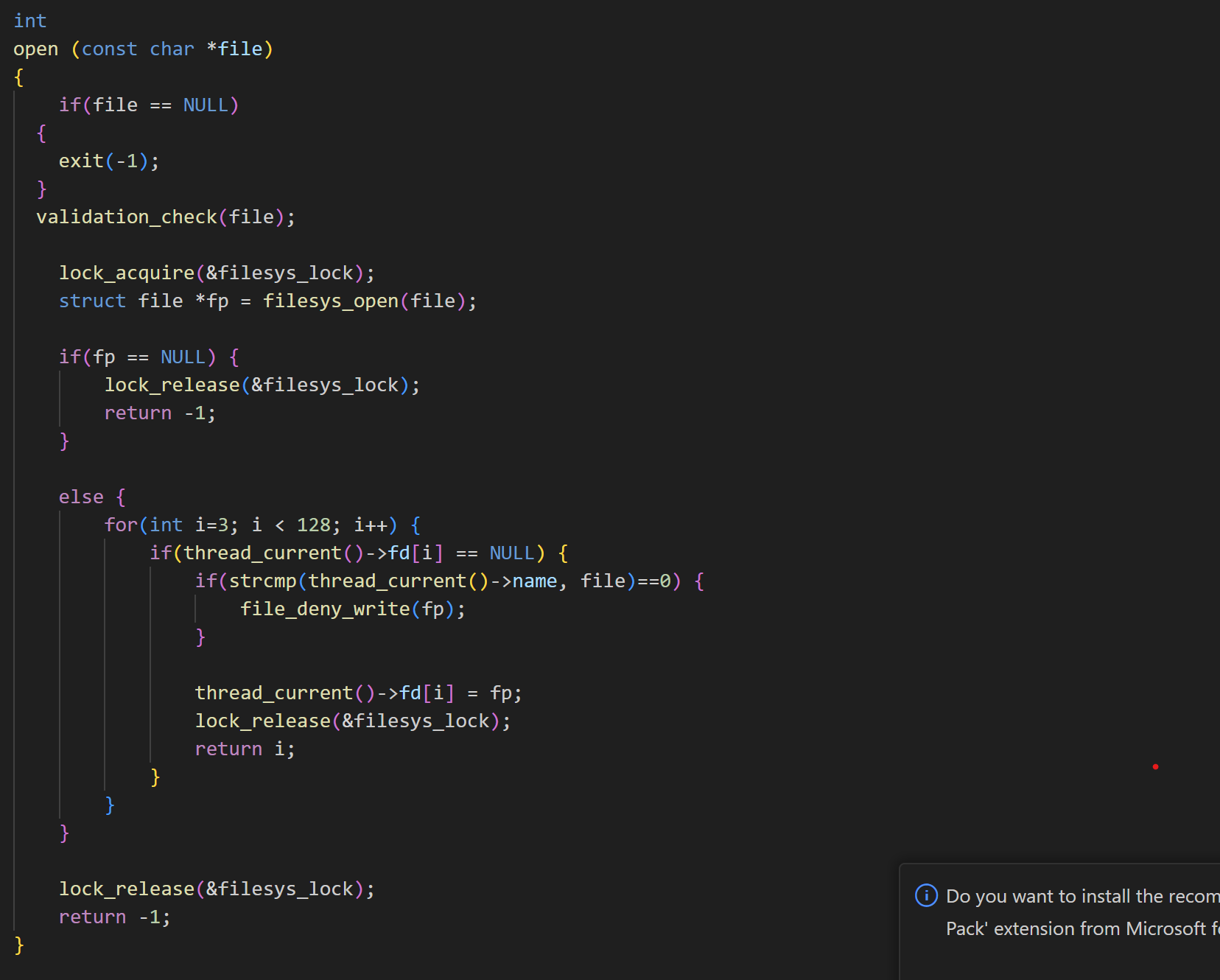
file == NULL 또는 validation\_check() 를 통해 유효하지 않은 경우 exit(-1) 로 종료한다.

create함수는 filesys\_create()함수를 호출하고 그 결과를 반환한다.

****

file== NULL 또는 validation\_check()를 통해 유효하지 않은 경우 exit(-1)로 종료한다.

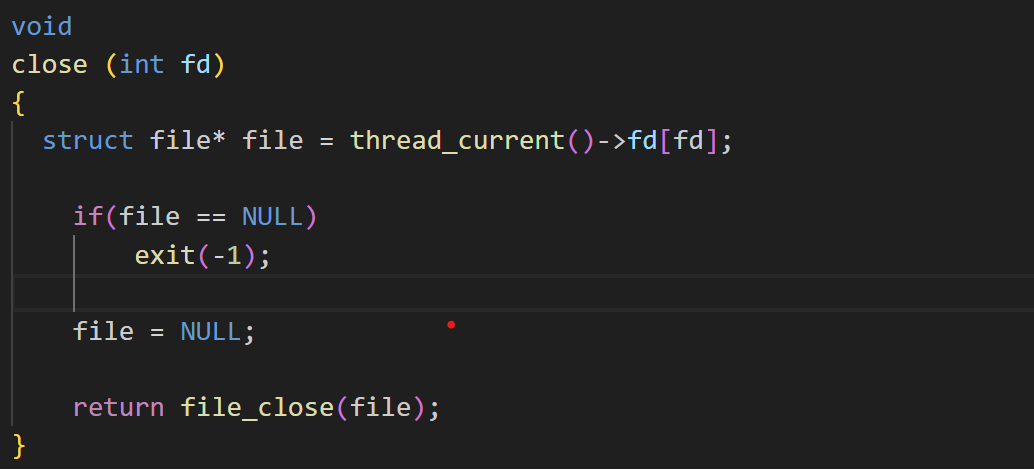
remove 함수는 filesys\_remove()함수를 호출하고 그 결과를 반환한다.

****

SYS\_OPEN에서 open 함수를 호출한다.

file== NULL 또는 validation\_check()를 통해 유효하지 않은 경우 exit(-1)로 종료한다.

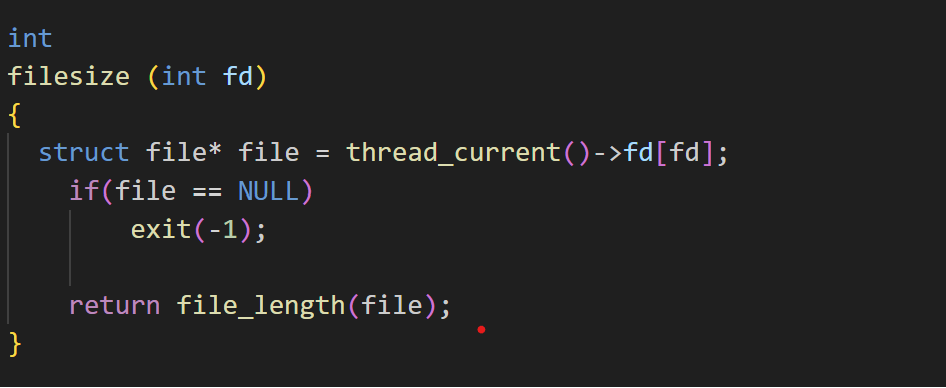
동기화 문제를 해결하기 위해 lock\_acquire, lock\_release를 사용한다.fp== NULL인 경우에는 lock\_release를 하고 -1을 반환한다. i=3부터 순회하면서 fd가 null인 경우 그 자리에 fp 값을 넣고 lock\_release 후에 그 fp값을 반환한다. 그 이전에 thread\_current의 name변수와 file변수를 비교하여 그 값이 같으면 file\_deny\_write을 한다.

****

SYS\_CLOSE에서 close 함수를 호출한다.

현재 fd가 NULL인 경우 exit(-1) 을 한다.

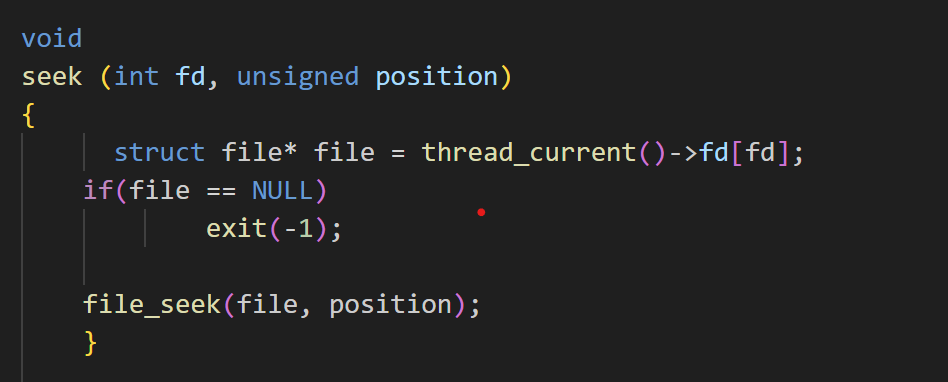
추가로 file에 현재의 thread의 fd값을 저장하고 해당 fd를 NULL로 바꾼다. 마지막으로 fileclose(file)을 호출하고 그 결과를 return 한다.

****

SYS\_FILESIZE에서 file size함수를 호출한다.

현재 thread의 fd가 NULL인 경우에는 exit(-1) 을 한다.

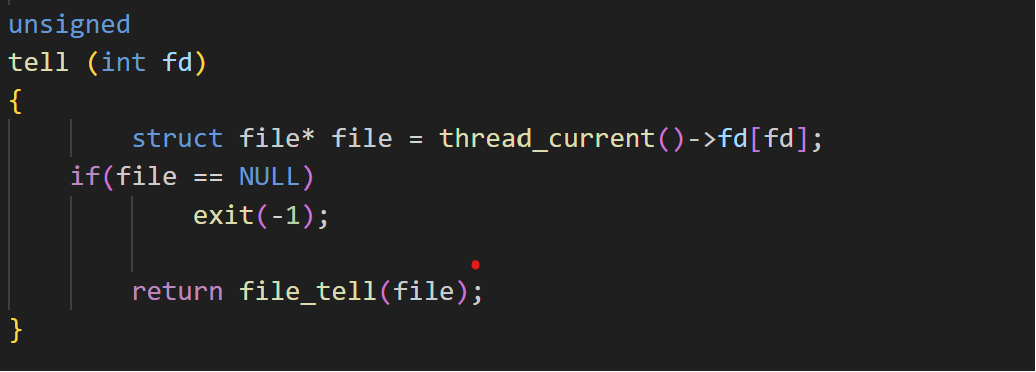
file\_length(thread\_current()->fd[fd])함수를 호출하고 그 결과를 return 한다.

****

SYS\_SEEK에서 seek을 호출한다.

현재 thread의 fd 가 NULL이면 exit(-1)을 한다.

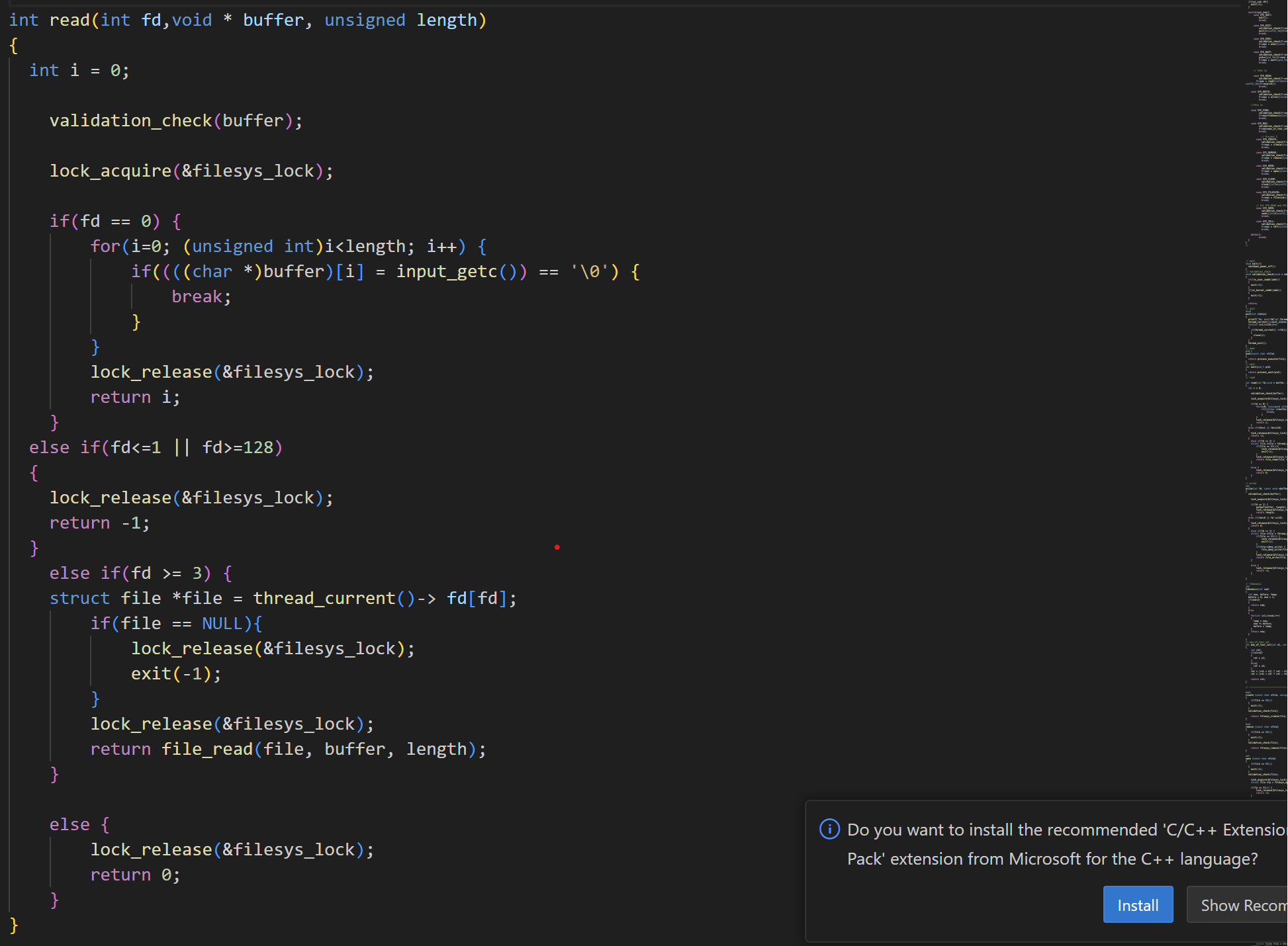
seek함수는 file\_seek(file,position)을 호출하고, 그 결과를 return 한다.

****

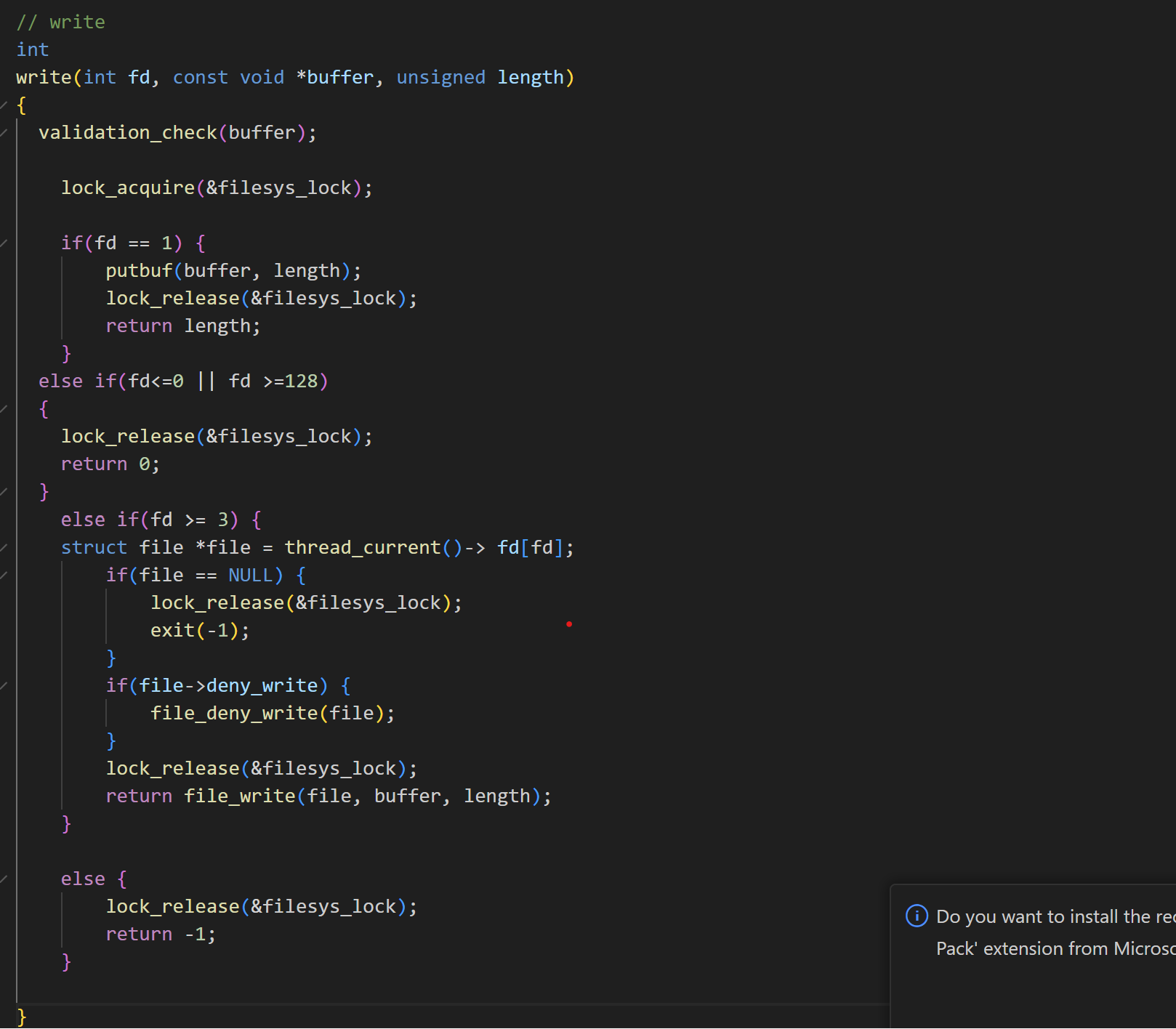
SYS\_TELL 에서 tell함수를 호출한다.

현재 쓰레드의 fd가 NULL인 경우 exit(-1)한다.

file\_tell(file)함수를 호출하고 그 결과를 return 한다.

****

stack을 통해 넘겨받은 fd를 인자로 받아 파일에 접근한다. 이때 해당 파일이 없으면 exit()을 호출하고, 있을 경우 STDIN인지 파일 입력인지 구분한다. STDIN인 경우에는 input\_getc() 함수를 사용해 buffer에 입력을 저장한다. 파일 입력일 경우 file\_read() 함수를 통해 파일에 저장된 값을 buffer에 저장한다. 시작시 lock\_acquire, 종료시 lock\_release를 통해 synchronization문제를 해결한다. 그리고 file\_read로 읽어들인 byte수를 반환한다.

****

stack을 통해 넘겨받은 fd를 인자로 받아 파일에 접근한다. 이때 해당 파일이 없으면 exit()을 호출하고, 있을 경우 STDOUT인지 파일 출력인지 구분한다. STDOUT인 경우에는 putbuf함수를 사용해 buffer에 출력을 저장한다. 파일 출력일 경우 file\_write() 함수를 통해 파일에 저장된 값을 buffer에 저장한다. 시작시 lock\_acquire, 종료시 lock\_release를 통해 synchronization문제를 해결한다. 그리고 file\_write로 출력한 byte수를 반환한다.

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

