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

담당 교수 / 분반 : 김영재 / 1반

이름 / 학번 : 한솔 / 20171705

개발 기간 : 2021.10.03 ~ 2021.10.15

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

file system을 유저프로그램에서 이용하기 위한 시스템 콜을 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

file을 오픈할 때, 각 파일을 유저프로그램에서 각 파일들을 이용하기 위한 핸들러로 file descriptor가 필요하다. 이를 통해서 write또는 read할 파일을 지정할 수 있다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

read : stdin에 대한 입력 뿐 아니라 모든 파일에 대한 입력을 처리할 수 있도록 구현한다.

write : stdout에 대한 출력 뿐만 아니라 모든 파일에 대해서 write할 수 있도록 구현해야한다.

create : 원하는 이름으로 파일을 생성하기 위해서 필요하다.

remove : 해당하는 파일을 삭제하기 위해서 필요하다.

open : 파일을 오픈하여 파일 디스크립터를 반환해줌으로써 이후 read나 write을 할 수 있게하기 위해서 필요하다.

filesize : 파일의 사이즈를 알아내는 데에 필요하다.

seek : 파일의 특정위치를 지정해서 읽거나 쓸 때 필요하다.

tell : 파일이 현재 어느부분을 읽거나 쓰는지 확인할 때 필요하다.

close : 오픈한 파일을 다시 닫을 때 사용되고, 이를 통해 파일을 읽거나 쓸 수 없게된다.

3. Synchronization in Filesystem

파일을 읽고 쓸 때, 이미 읽히고 있는 파일에 쓸 수 없게하고 이미 쓰여지고 있는 파일을 읽을 수 없게한다. 또한 쓰고있는 파일에 쓰는 것도 불가능하다. 하지만 읽고 있는 파일을 읽는 것은 가능해야한다. 이를 위해 read와 write을 할 때 동기화가 필요하다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

struct list\_elem를 이용해서 파일디스크립터와 파일들을 linked list로 연결한다. 어떤 파일디스크립터가 사용되고 있는지 파악하기 위해서 bitmap자료구조를 이용한다. 사용되는 파일디스크립터에 해당하는 bitmap을 set한다.

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)

read : 파일디스크립터를 입력받아서 이에 해당하는 파일로부터 원하는 사이즈만큼 읽어서 버퍼에 저장한다. 반환값은 읽은 사이즈이다. 파일디스크립터가 0인 경우는 stdin에 해당한다.

write : 파일디스크립터를 입력받아서 이에 해당하는 파일을 사이즈만큼 버퍼에 입력한다. 반환값은 입력한 사이즈가 된다. 파이리스크립터가 1인 경우는 stdout에 해당한다.

create : 파일명을 입력받아서 파일을 생성할 수 있는 시스템 콜으로 원하는 이름의 파일을 생성할 수 있게된다.

remove : create와 반대로 입력받은 파일명에 해당하는 파일을 삭제하는 기능을 수행한다.

open : 파일명을 입력받아서 그에 해당하는 파일을 오픈하고, 파일 디스크립터를 반환한다. read, write등을 수행하기 위해서 파일을 오픈하고 이를 지정할 수 있게 해준다.

filesize : 파일디스크립터를 입력받아서 파일의 사이즈를 반환한다.

seek : 파일디스크립터와 원하는 위치를 입력받아서 오픈한 파일의 어느부분을 읽거나 쓰고있는 지를 나타내는 값을 원하는 값으로 변경한다

tell : 파일디스크립터를 전달받아서 파일의 어느 위치를 읽거나 쓰고있는지를 반환한다.

close : 파일디스크립터에 해당하는 오픈한 파일을 닫는다. 해당 파일에 읽거나 쓸 수 없게된다.

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

semaphore를 이용한 방법으로는, write 수행전에 sema\_down 수행후에 sema\_up을 사용한다. read의 경우에는 여러 프로그램이 동시에 read할 수 있어야하므로 처음 read할 때만 sema\_down을 사용하고, 마지막으로 read를 끝냈을 때 sema\_up을 사용한다.

lock을 이용한 방법은, write을 수행하기 전에 lock\_aquire를 실행하고, 수행한 후에 lock\_release를 실행한다. 그리고 read의 수행전에 lock\_aquire를 실행하고, 수행 후에 lock\_release를 실행한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

**10. 03 ~ 10. 05 create, remove system call구현**

**10.06 ~ 10. 08 open systemcall을 구현하면서 file descriptor를 함께 구현**

**10.09 ~ 10.10 구현된 file descriptor를 이용해서 나머지 system call 중 filesize, seek, tell, close를 구현.**

**10.11 ~ 10.12 read와 write system call을 stdin, stdout뿐 아니라 모든 파일에 대해서 확장.**

**10.13 ~ 10.15 read와 write할 때 synchronization을 구현.**

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

userprog/syscall.c에 create,remove,open,filesize,seek,tell,close 시스템콜을 추가한다.,

* + 수정하거나 추가해야 하는 자료구조

threads/thread.h 에서 struct thread의 멤버에 struct list와 struct bitmap\*를 추가해야한다. 파일을 저장하고, 사용중인 파일 디스크립터를 확인하기 위한 용도이다.

lib/kernel/list.h에서 struct list\_item을 추가한다. 이는 멤버로 struct list\_elem, int fd, struct file\*를 가진다. 파일과 파일디스크립터를 저장하는 노드에 해당한다.

* + 수정하거나 추가해야 하는 함수

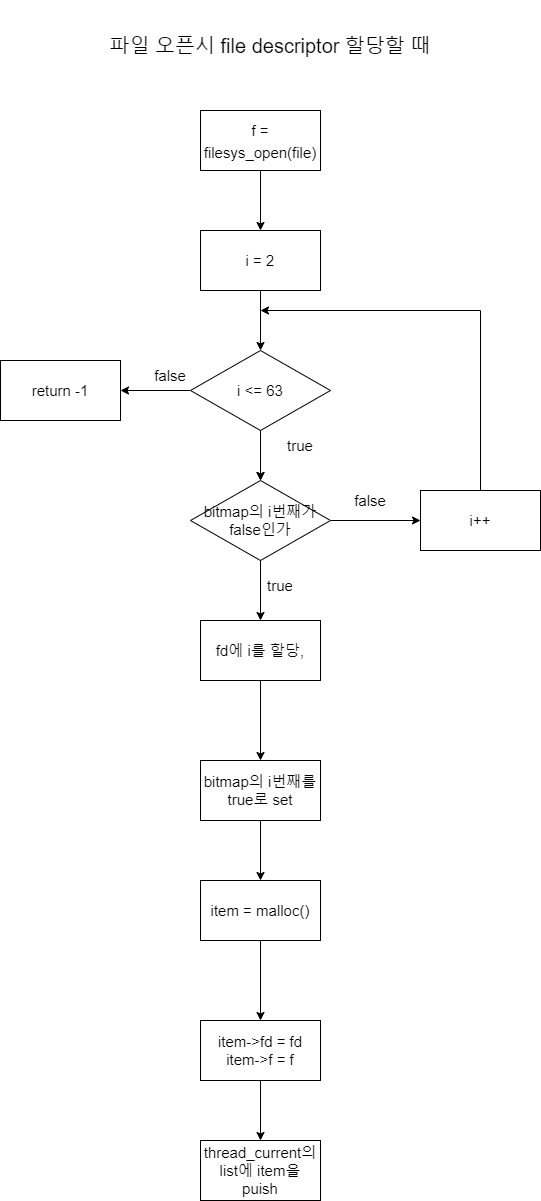
userprog/syscall.c에서 read,write를 수정한다. read, write은 stdin,stdout 뿐아니라 모든 파일에 대해서 읽고 쓰기를 수행할 수 있어야한다.

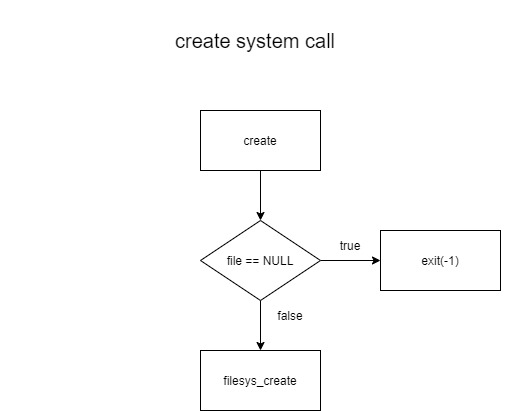
filesys/file.c에서 file\_write\_sema\_down, file\_write\_sema\_up, file\_read\_sema\_down, file\_read\_sema\_up을 추가한다. 이는 각각 write시스템콜에서 파일을 입력받아서 sema\_down, sema\_up을 해주는 함수, read시스템콜에서 파일을 입력받아서 sema\_down, sema\_up을 해주는 함수이다.

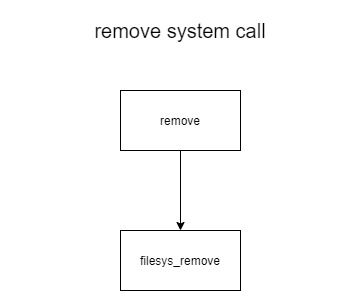
threads/thread.c에서 bool thread\_findname\_foreach(const char\* name)을 추가해야 한다. 이는 전체 thread중에서 입력받은 이름에 해당하는 thread가 있을시 true를 반환하고 없을 시에 false를 반환한다.

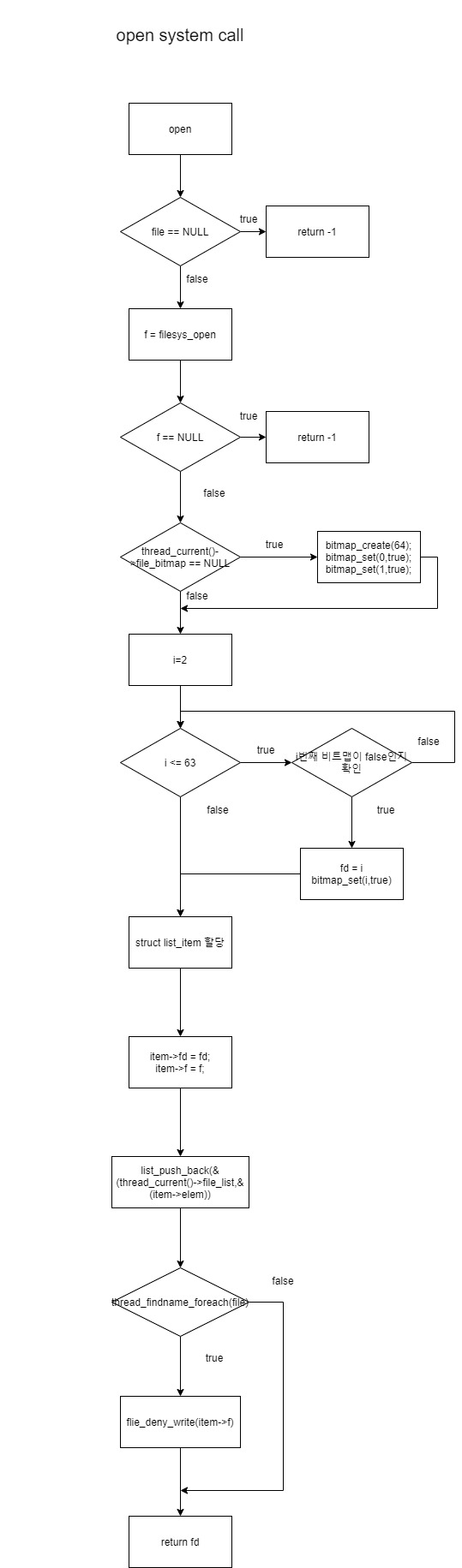
1. **연구 결과**
   1. **Flow Chart**

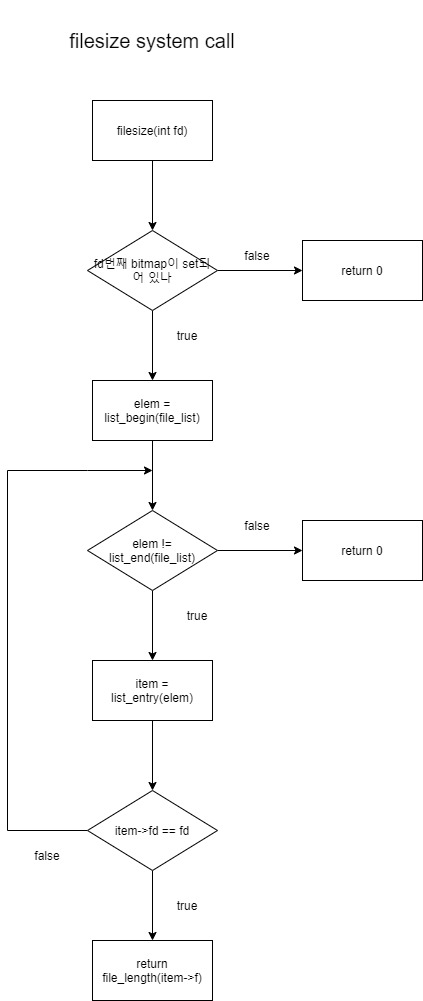
* II. B. 개발 내용의 각 3가지 항목에 대하여 Flow Chart 작성

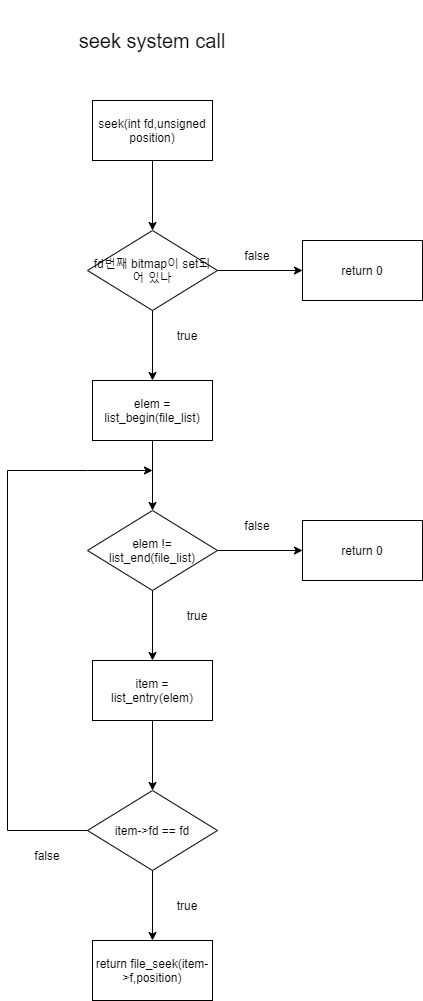


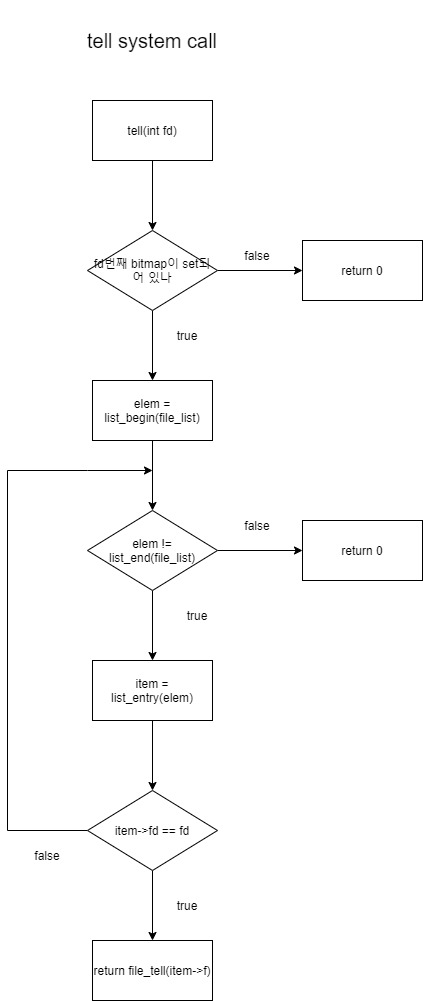
****

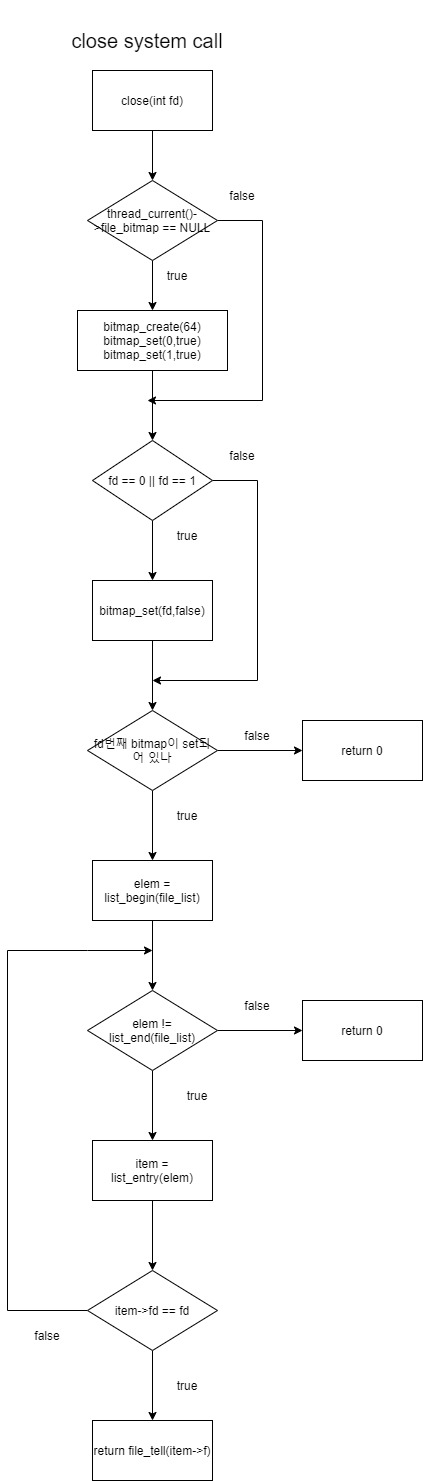
****

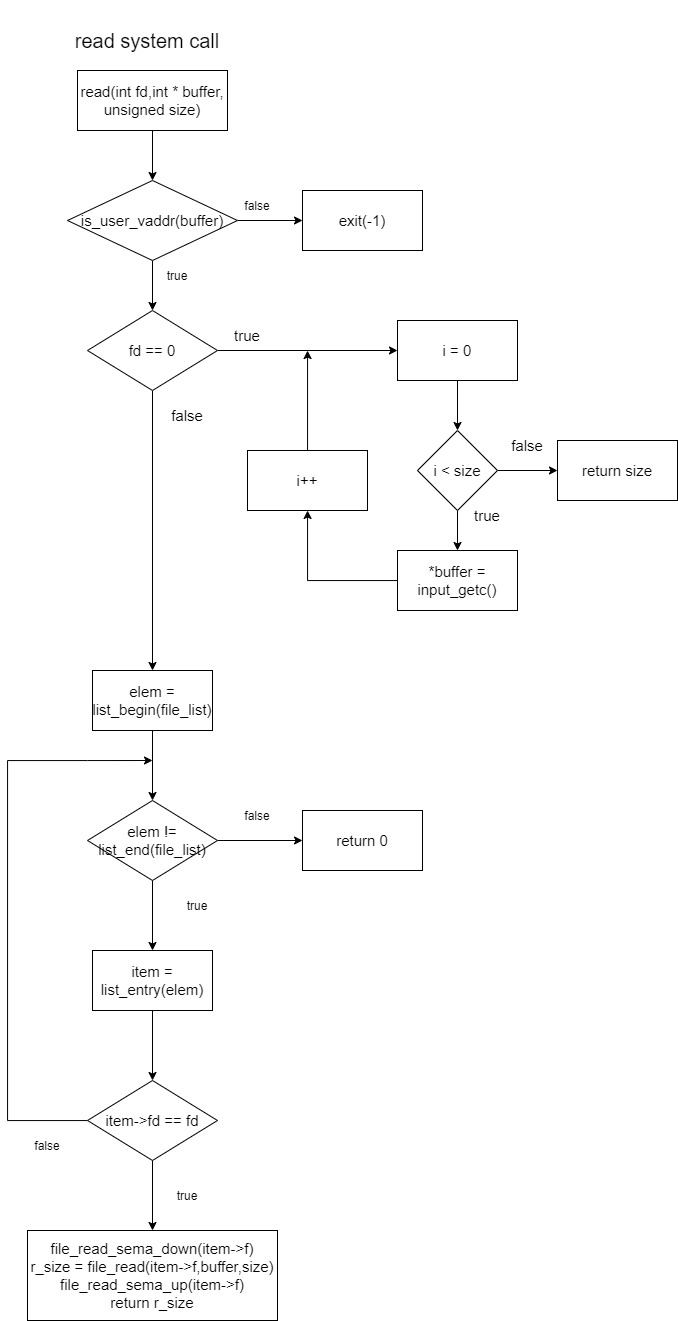
****

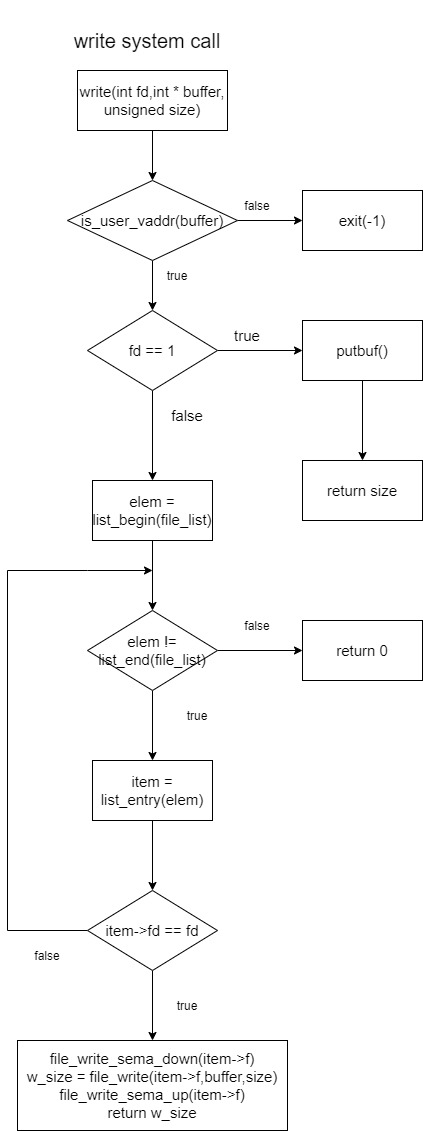
****

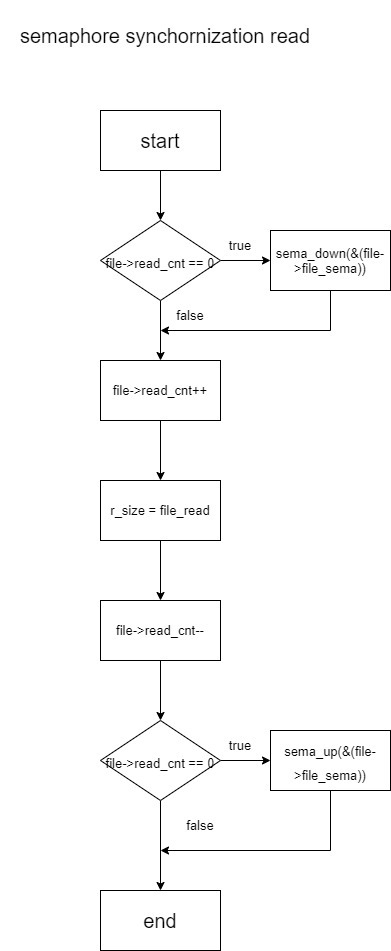
****

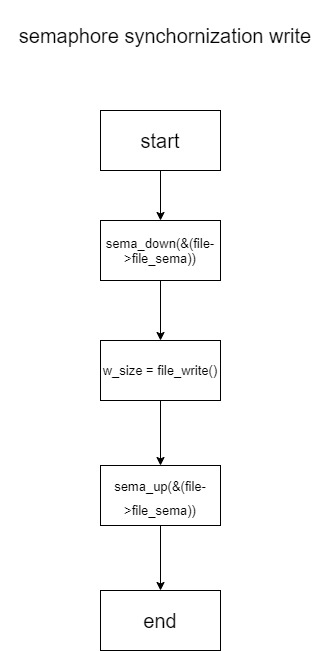
****

****

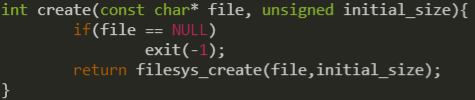
****

****

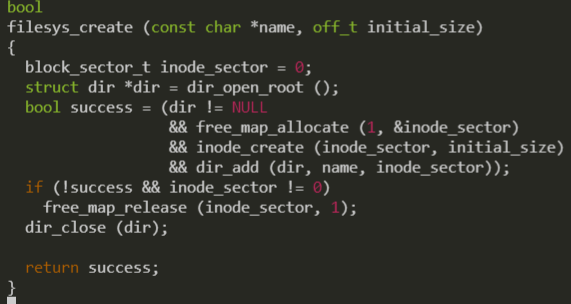
****

****

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

****

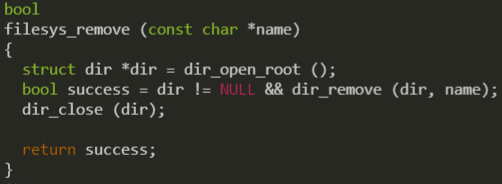
**create 시스템 콜을 구현하기 위해서 create라는 함수를 만들었고, 입력된 filename이 NULL이 아닌 경우에 filesys\_create라는 내장함수를 호출한다.**

****

**filesys\_create함수는 root directory를 오픈해서 파일을 저장할 공간을 할당하고, inode를 생성한다. 그리고 dir에 name을 가진 파일을 추가한다. 이에 실패하면 false를 반환하고, 성공하면 true를 반환한다.**

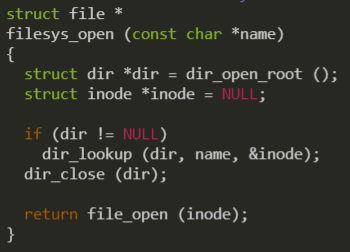
****

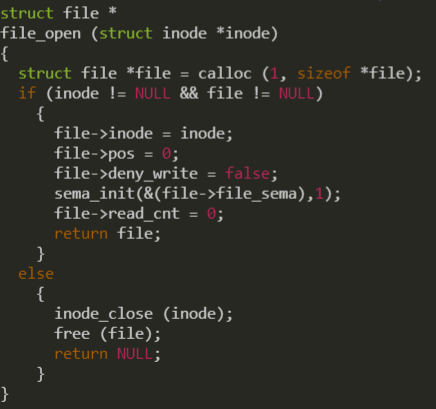
**remove 시스템 콜을 구현하기 위해서 remove라는 함수를 만들었다. 이 함수는, filesys\_remove라는 내장함수를 호출한다.**

****

**filesys\_remove함수는 root directory를 오픈해서 dir에 저장하고 전달받은 name에 해당하는 파일을 dir에서 삭제한다.**

****

****

****

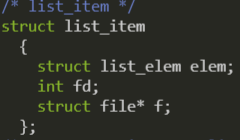
**open 시스템콜을 위해서 open함수를 구현하였다. 이는 입력받은 파일이름이 NULL이라면 -1을 반환한다. 그렇지 않은 경우에는 filesys\_open이라는 내장함수를 호출한다. 이 내장함수는 dir를 오픈해서 입력받은 name에 해당하는 inode를 찾는다. 찾았다면 file\_open이라는 내장함수를 호출한다. file\_open함수는 새로운 파일 구조체를 할당받아서 구조체의 멤버를 초기화한 후에 반환해준다. open함수로 돌아와서 만약 NULL을 반환받은 경우에는 -1을 리턴한다.**

****

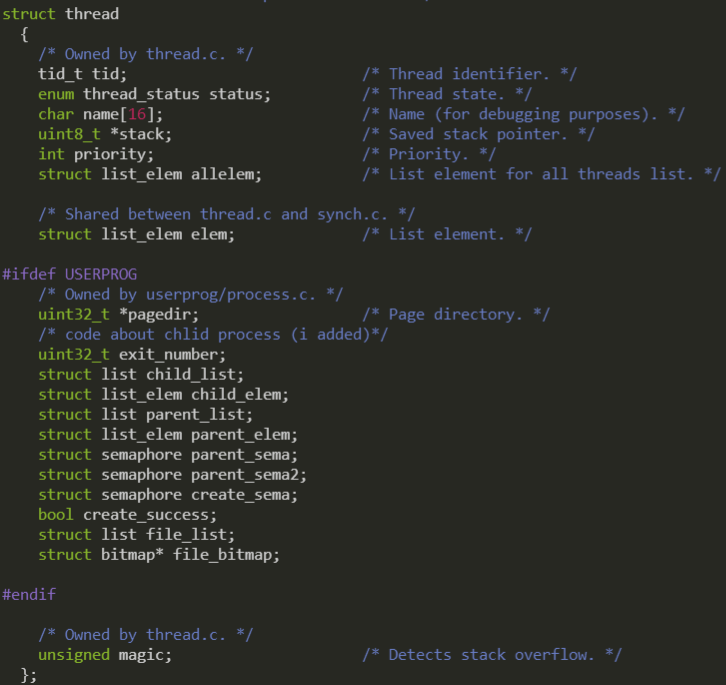
**open함수와 close함수에서는 현재 스레드의 bitmap이 아무것도 할당되어있지 않은 경우에 비트맵을 생성하고 0,1을 stdin,stdout을 위해 true로 set해준다.**

****

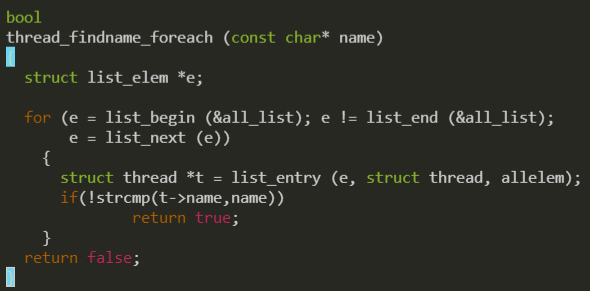
**파일 디스크립터를 할당해주기 위해서 반복문을 통해서 비트맵의 가장 작은 값의 파일디스크립터를 찾아서 true로 set해주고 해당 index를 파일디스크립터로 사용한다. 그리고 linked list의 node에 해당하는 list\_item을 할당하고 파일과 파일디스크립터를 멤버변수로 초기화해준 후, 리스트를 push한다. 모든 thread의 list인 all list에서 오픈하려는 이름을 가진 파일이 이미 실행중인지 확인하기 위해서 구현한 thread\_findname\_foreach라는 함수를 사용한다. 실행중이라면 file\_deny\_write를 사용해서 write할 수 없도록 한다. 그리고 파일디스크립터를 반환하고 종료한다.**

****

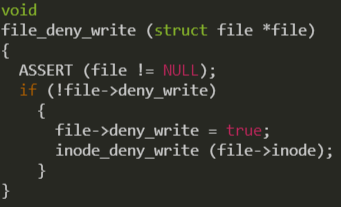
**list\_item 구조체는 위와 같고, list\_elem과 파일디스크립터를 저장하는 fd, file을 저장하는 f로 이루어져 있다.**

****

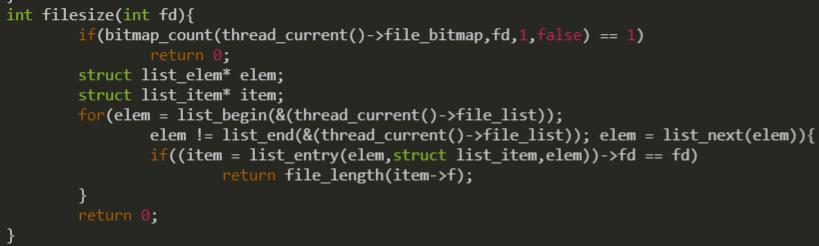
**thread 구조체에 struct list와 bitmap을 추가했다. list는 파일들과 그 파일에 대응되는 파일 디스크립터를 linked list로 관리하기 위함이고, bitmap은 사용중인 파일 디스크립터를 확인하기 위함이다.**

****

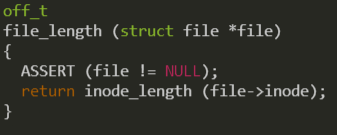
**위는 userprog/thread.c에 새롭게 구현한 함수이다. 이는 all list의 모든 노드를 확인해서 입력받은 name에 해당하는 스레드가 all list에 존재하는지 확인한다. 존재한다면 true를 아니라면 false를 반환한다.**

****

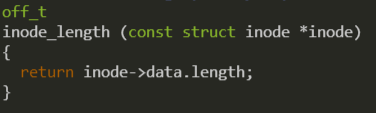
**위는 내장함수인 file\_deny\_write이다. 파일 구조체의 deny\_write값을 true로 변경하는데, 이렇게하면 이후 write을 할 수 없게된다.**

****

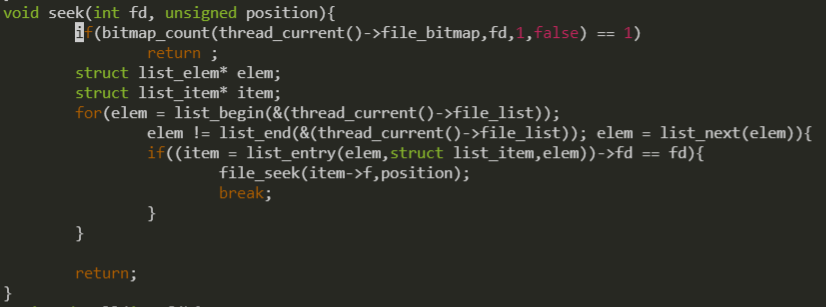
**위는 filesize시스템콜을 구현하기 위한 filesize함수이다. 이는 시스템콜을 호출한 현재 스레드의 리스트의 노드들을 따라가며 입력받은 파일디스크립터에 해당하는 값을 가진 노드를 찾으면 해당파일에 대해서 file\_length라는 내장함수를 실행한다.**

****

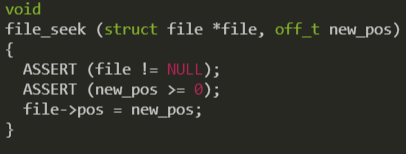
**file\_length는 다음과 같이 inode\_length라는 내장함수를 실행하는데,**

****

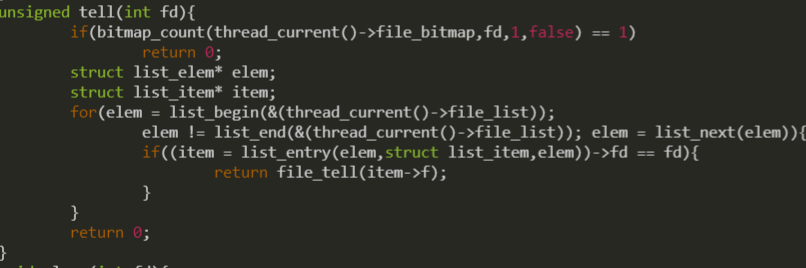
**이는 입력받은 inode의 크기를 반환해준다. 즉 파일의 크기를 반환한다.**

****

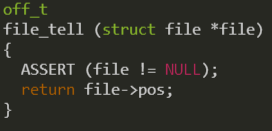
**위는 seek 시스템콜을 위해서 구현한 seek함수이다. seek함수는 시스템콜을 호출한 현재 스레드의 리스트의 노드를 이동하면서 입력받은 파일디스크립터에 해당하는 값을 가진 노드를 찾는다. 그 노드가 가리키는 파일에 대해서 file\_seek를 호출한다.**

****

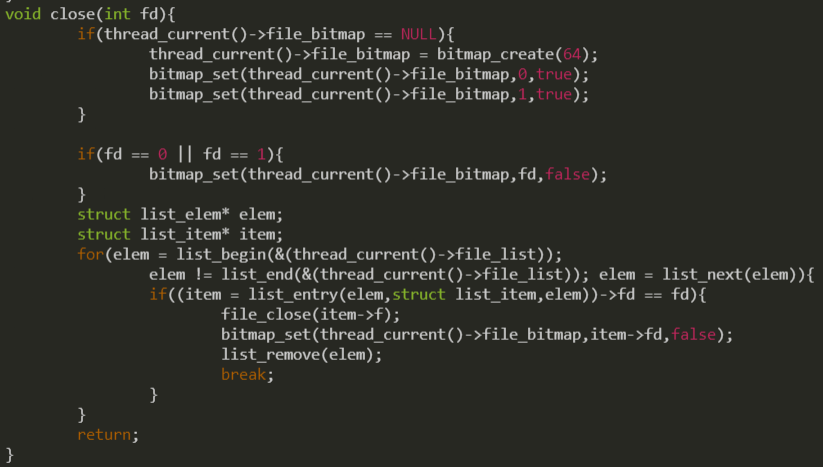
**file\_seek는 위와 같은 내장함수인데, 파일이 입력받은 새로운 위치를 가리키도록 한다.**

****

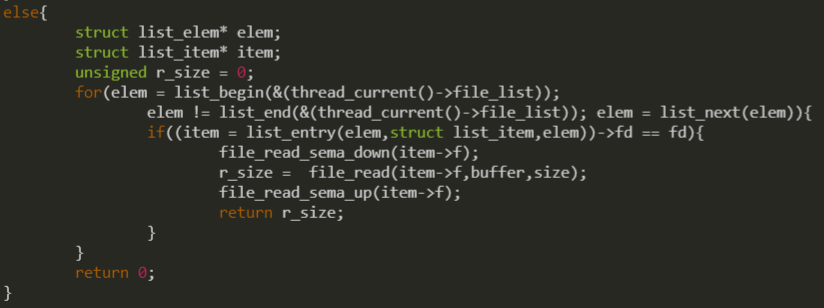
**위는 tell 시스템콜을 구현하기 위한 tell함수이다. 이 함수는 시스템콜을 호출한 스레드의 파일 리스트의 노드를 따라가며 입력받은 파일디스크립터에 해당하는 값을 가진 노드를 찾는다. 그리고 그 노드가 가리키는 파일을 입력으로 file\_tell내장 함수를 호출하고 그 반환값을 반환한다.**

****

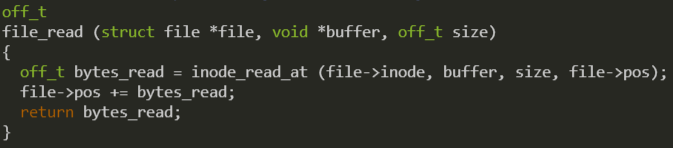
**file\_tell함수는 파일을 입력받아서 현재 파일이 가리키는 위치, 즉 읽거나 쓴 후의 최종 위치를 반환해준다.**

****

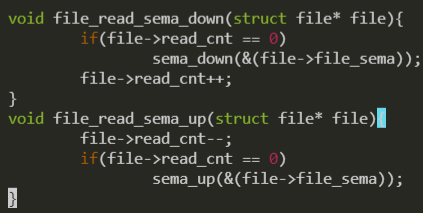
**위는 close 시스템콜을 구현하기 위한 close함수이다. 이는 open과 마찬가지로 시스템콜을 호출한 스레드의 file\_bitmap에 아무것도 할당되지 않았을 시에, 할당해주고 0,1을 true로 set해준다. 입력받은 fd가 0이거나 1인 경우 이에 해당하는 비트맵을 false로 바꾼다. 아닌 경우에는 스레드의 파일리스트를 탐색하며 fd에 해당하는 노드를 발견할 시, 그 노드가 가리키는 파일을 close하고 그 fd를 bitmap에서 false로 바꾼다. 그리고나서 file list에서 제거한다.**

****

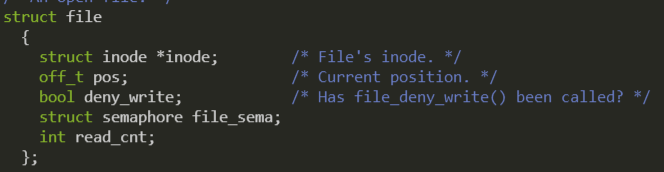
**위는 read 함수에서 fd가 0이 아닐 때에 해당하는 코드이다. 시스템콜을 호출한 스레드의 파일 리스트에서 입력받은 파일디스크립터에 해당하는 노드를 찾은 후 해당 노드가 가리키는 파일에 대해서 file\_read함수를 호출한다. 이때 synchronization을 위해서 호출전에 file\_read\_seam\_down함수, 호출후에 file\_read\_sema\_up함수를 사용한다.**

****

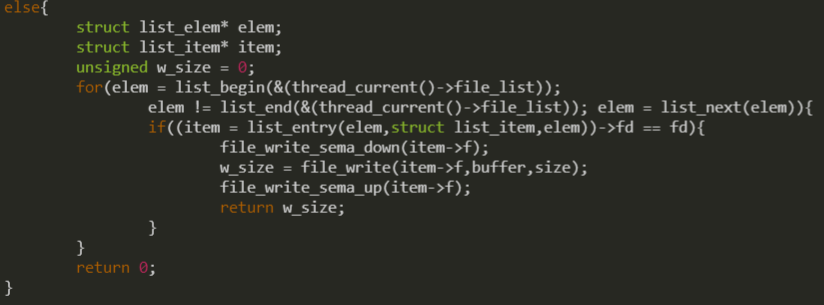
**file\_read 내장함수는 위와 같이 입력받은 파일의 inode를 사용하여 inode\_read\_at을 호출하여 읽은 바이트의 크기를 반환한다.**

****

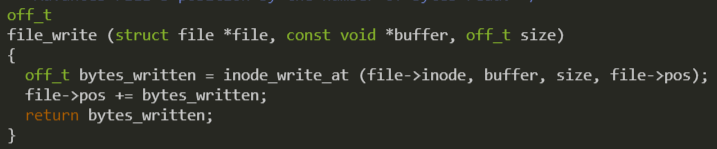
**위는 filesys/file.c파일에 구현한 file\_read\_sema\_down과 file\_read\_sema\_up이다. file\_read\_sema\_down은 파일을 여러 개의 프로그램에서 읽을 수 있도록 해준다. 하지만 read도중에 write하거나 write도중에 read하는 것은 불가능해야하므로, 첫번째로 read를 할때만 sema\_down을 한다. 이를 위해서 read\_cnt를 이용한다. 이는 현재 파일을 읽고 있는 수를 의미한다. file\_read\_sema\_up은 마찬가지로 마지막으로 read를 실행한 프로그램에서만 sema\_up을 한다. 이렇게 해서 가장 처음 read를 할때와 마지막으로 read를 끝냈을 때만 sema up, down을 실행하게 된다.**

****

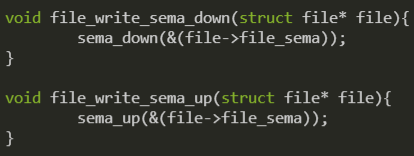
**위는 struct file에 새로 추가한 멤버들이다. struct semaphore는 synchronization 을 위해 추가했고, read\_cnt또한 얼마나 많은 수가 파일을 읽고 있는 지를 나타내기 위해서 추가했다.**

****

**위는 write함수의 fd가 1이 아닌 경우에 해당하는 코드이다. 시스템콜을 호출한 스레드의 파일 리스트를 탐색하여 입력받은 파일디스크립터에 해당하는 노드를 찾는다. 그리고 그 노드가 가리키는 파일 argument로 file\_write를 호출한다. file\_write의 호출 전과 후에는 각각 file\_write\_sema\_down과 file\_write\_sema\_up을 실행해서 누군가 읽거나 쓰는 중에 쓸 수 없도록 synchronization해준다.**

****

**file\_write 내장함수는 위와 같다. 입력받은 파일의 inode를 이용해서 inode\_write\_at을 호출하고 이로부터 write한 바이트 수를 반환받는다. 그 값만큼 파일의 pos즉 위치를 나타내는 변수를 업데이트 해주고, 쓴 바이트를 반환한다.**

****

**file\_write\_sema\_down과 file\_write\_sema\_up 파일을 입력받아서 그 파일의 file\_sema를 이용해서 sema\_down과 sema\_up을 수행해준다. 이는 syncrhonization을 위해서 사용한다.**

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

