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

담당 교수 / 분반 : 문의현 / 2분반

이름 / 학번 : 박효리 / 20191251

개발 기간 : 11/1-11/14

1. **개발 목표**

이번 project2에서는 file system과 관련된 system call을 구현하는 것이 목적이다. 구현해야 할 것은 create, remove, open, close, filesize, read, write, seek, 그리고 tell이 될 것이다. API를 통해 이러한 system call을 구현함으로써 pintos의 file system을 구현하며, 이때 critical section을 위해 동기화 기능도 구현해준다.

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

1. file descriptor는 file에 접근할 때 사용되는 값으로, 특정 파일을 open할 때 커널은 file descriptor에서 NULL값을 갖는 가장 작은 값을 그 파일에 할당한다. 이 경우 또 다른 process가 open돼있는 같은 파일에 접근하려 할 때 이 file descriptor 값을 통해 접근한다. 따라서 우리는 file descriptor배열을 만들어주어야 할 것이다.

2. 이번 프로젝트에서 구현해야 할 것은 앞서 언급했듯 create, remove, open, close, filesize, read, write, seek, 그리고 tell이다. 지난 프로젝트와 같이 syscall\_handler내에서 system call들을 구현한다. 이때 API를 참조할 수 있다.

3.

여러 process는 data를 공유할 수 있고 이를 shared data라 한다. 또한 이 shared data에 접근하는 code section을 critical section이라고 부른다. 우리는 여러 process가 critical section에 접근하는 경우를 막아야 한다. 따라서 이를 위해 lock을 사용해서 동기화를 구현하여 동시에 여러 process가 한 critical section에 접근함을 막는다.

**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)

1. file descriptor는 file 구조체를 원소로 갖는 배열로 선택하였다. 이는 배열은 index를 통해 각 원소에 접근하기가 용이하기 때문이다. pintos는 한 process 당 최대 128개의 open file을 가질 수 있음을 manual에 언급하였기 때문에 file descriptor 배열의 크기를 128로 설정해준다.

2. create은 file명과 size를 입력 받아서 이를 가지고 파일을 만든다. 파일이 문제없이 만들어졌다면 true를, 그렇지 않다면 false를 return 한다. remove의 경우에도 file명을 입력 받아서 그 파일을 지운다. open은 파일명을 입력 받아서 그 파일을 열고, 파일을 여는데 실패했으면 -1을, 성공했다면 파일에 할당되는 file descriptor값을 반환해준다. close는 file descriptor을 입력 받아 그 파일을 닫아주고, 할당 받았던 file descriptor값을 다시 NULL로 하여 재사용되게 한다. filesize는 파일명을 입력 받아 해당 파일의 size를 반환한다. read는 file descriptor값과 buffer, size를 입력 받아 해당 파일을 size만큼 읽어 들여 buffer에 저장해준다. write 역시 file descriptor값과 buffer, size를 입력 받아 해당 파일에 buffer 내용의 size만큼을 해당 파일에 적어준다. seek은 file descriptor와 위치를 입력 받아 파일의 read나 write하는 시작점을 입력 받은 위치로 옮긴다. tell은 file descriptor를 입력 받아 해당 파일의 read나 write의 시작점을 반환한다.

3. lock의 경우 read, write, 그리고 open에서 사용된다. lock\_init함수로 lock을 초기화하고, lock\_acquire함수와 lock\_release함수를 통해 lock을 걸었다 풀었다 하면서 critical section을 보호한다. semaphore의 경우, sema\_init으로 초기화를 하고, sema\_down과 sema\_up함수를 사용해서 동기화를 구현한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

**11/1-11/3: 프로젝트 내용 정리**

**11/4-11/8: file descriptor 선언과 system call 구현**

**11/9-11/13: 동기화 구현, 디버깅**

**11/4: 문서 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 descriptor배열을 선언하기 위해 file.c에 있는 file구조체를 file.h로 옮기고, thread.h에서 thread구조체 안에 fd라는 이름의 file 구조체 배열을 선언한다. 또한 thread.c의 init\_thread함수에서 for문을 통해 배열의 원소를 NULL로 초기화한다.

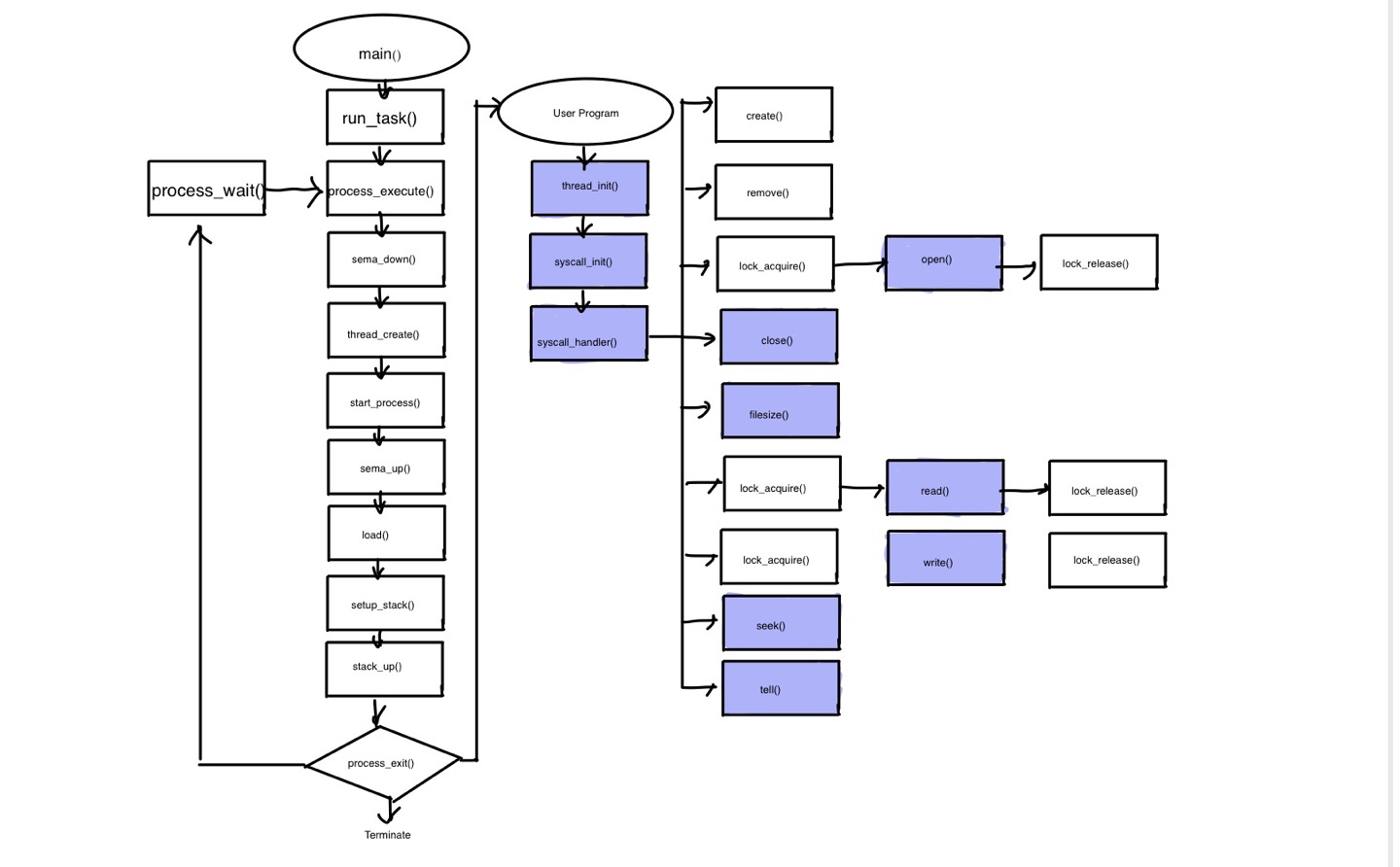
2. syscall.c에서 create, remove, open, close, filesize, read, write, seek, tell을 syscall handler안에서 구현해준다. 이때 필요한 함수들은 추가로 선언하고 작성한다.

3. thread.h에서 filesystem의 동기화를 위해 filesys\_lock이라는 이름의 lock을 선언한다. 그 다음, file process의 execution상태를 확인하는 set\_wait변수와 부모를 가리키는 thread구조체 포인터, file system에서 사용될 filesys semaphore, 그리고 현재 보고 있는 file을 저장하는 file구조체 포인터 변수들을 thread구조체 내에서 선언해준다. 이들은 init\_thread에서 초기화가 이루어진다. 그 다음, exception.c 파일 내의 page\_fault함수 내에서 not\_process가 true인 경우 exit을 호출하는 부분을 추가해준다. process.c 파일 내에서는 process\_execute 함수와 start\_process 함수에서 process가 작업을 안전하게 끝낸 경우나, process가 load되기 전에 부모 process가 끝나버리는 경우에만 메모리 정리가 되게끔 해주는 코드를 추가한다. load함수에서는 마지막 부분에 success가 true인 경우에 file\_deny\_write함수를 호출하여 write를 막는 코드를 추가해준다. 마지막으로 syscall.c에서 lock 관련 함수들과 filesys\_lock을 통해 동기화를 구현해준다.

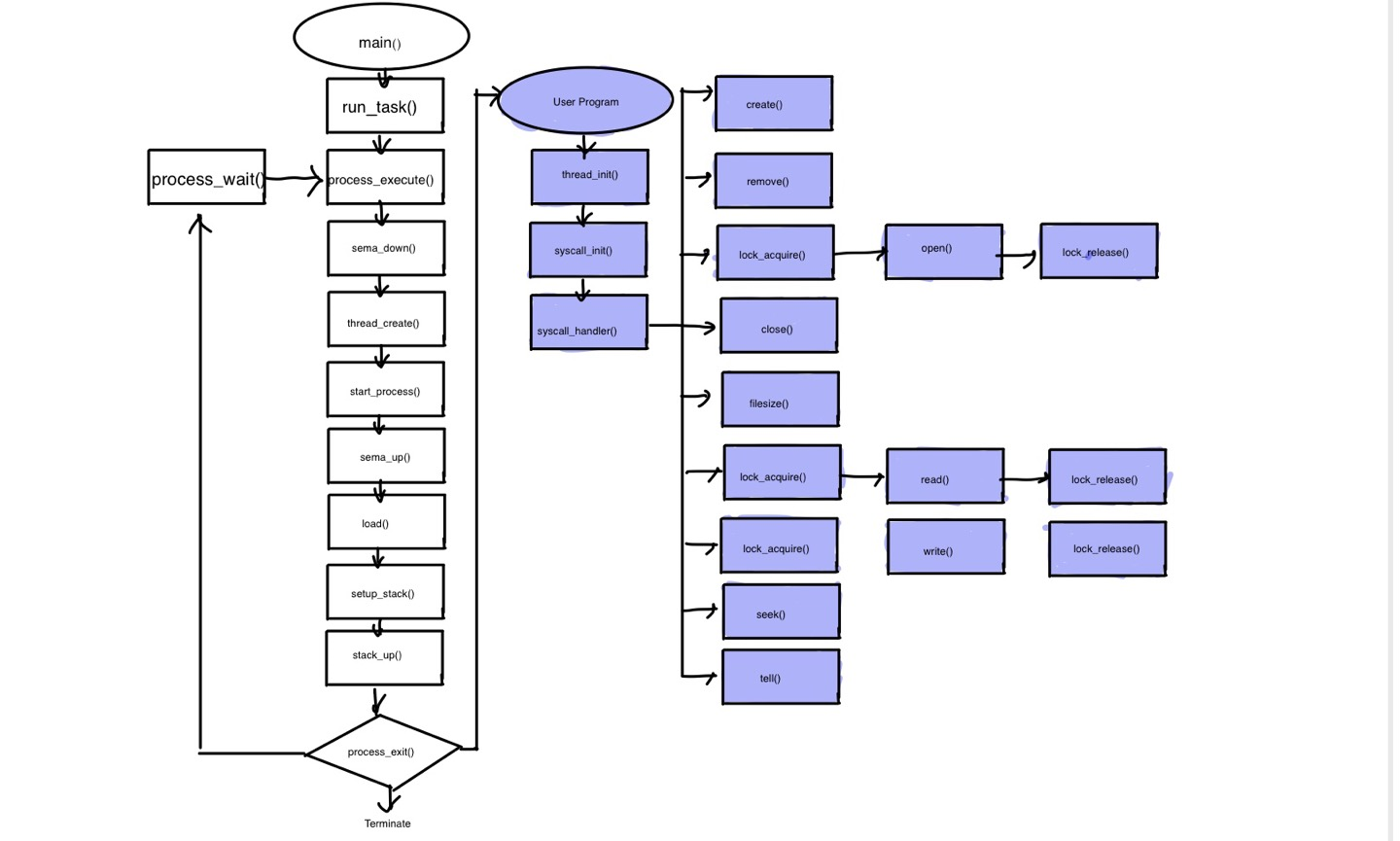
1. **연구 결과**
   1. **Flow Chart**

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

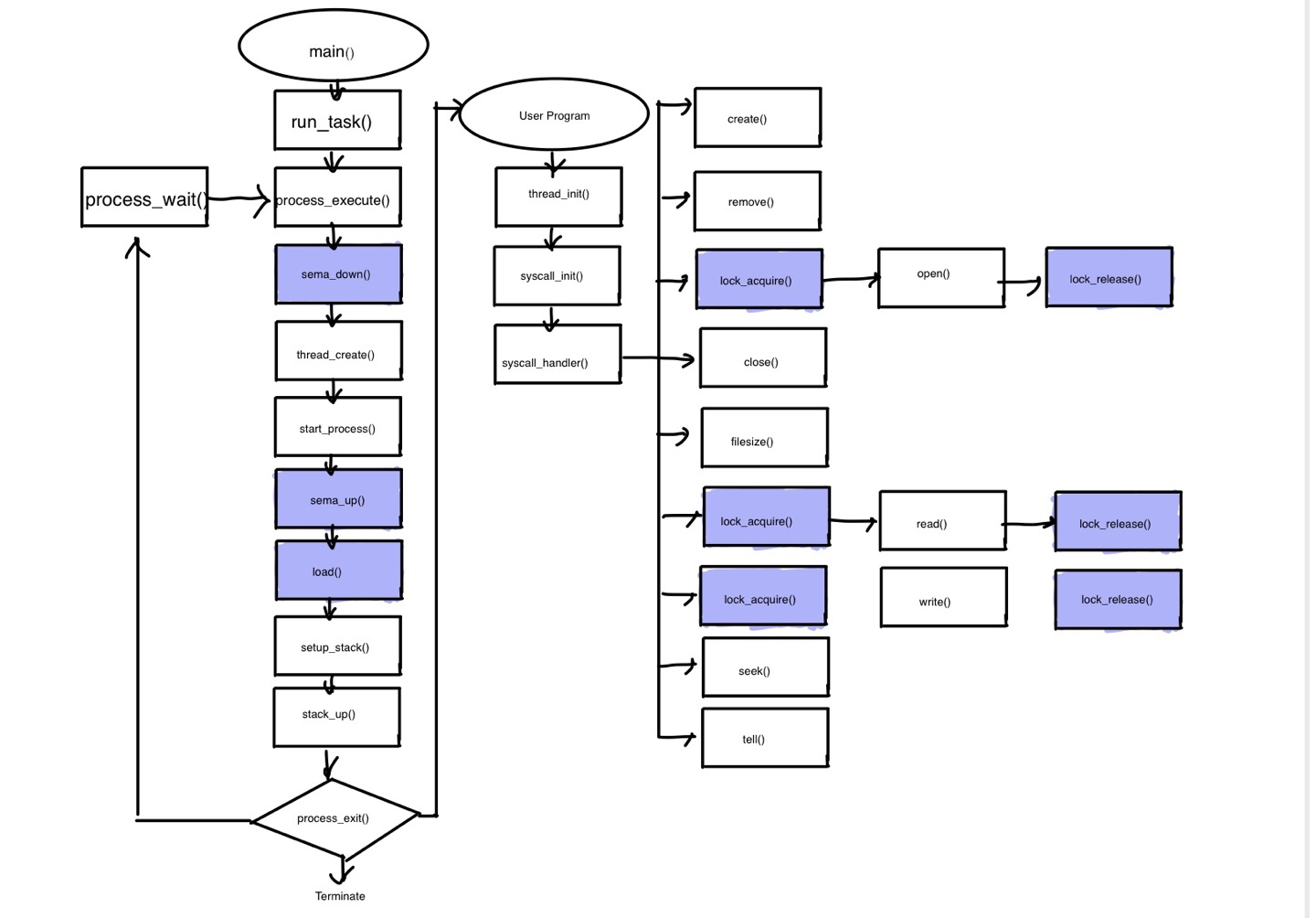
**1.**



**2.**

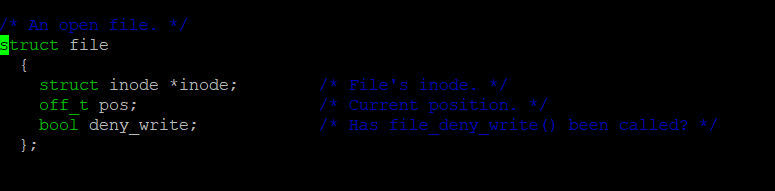


**3.**

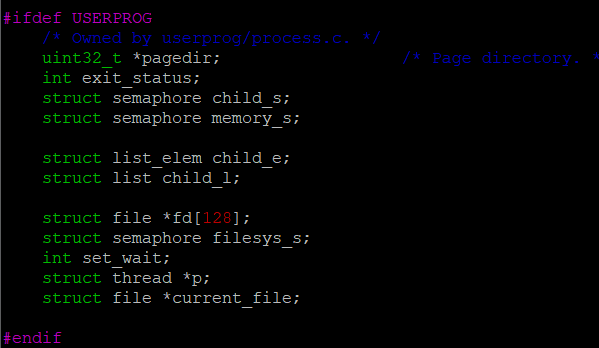


* 1. **제작 내용**
* II. B. 개발 내용의 각 3가지 항목에 대하여 실직적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
* 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명

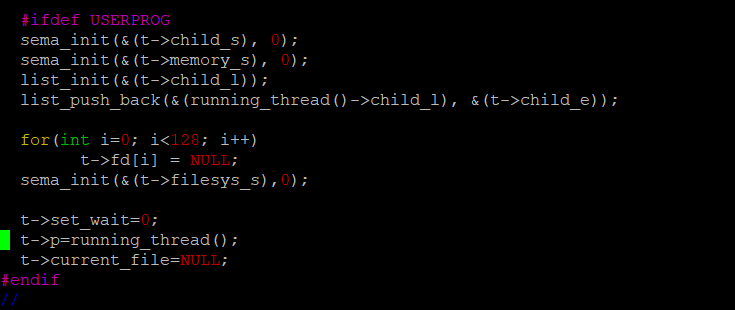
1.



file 구조체를 file.c에서 file.h로 옮겨 다른 파일에서도 이 구조체에 접근이 가능하게 한다.

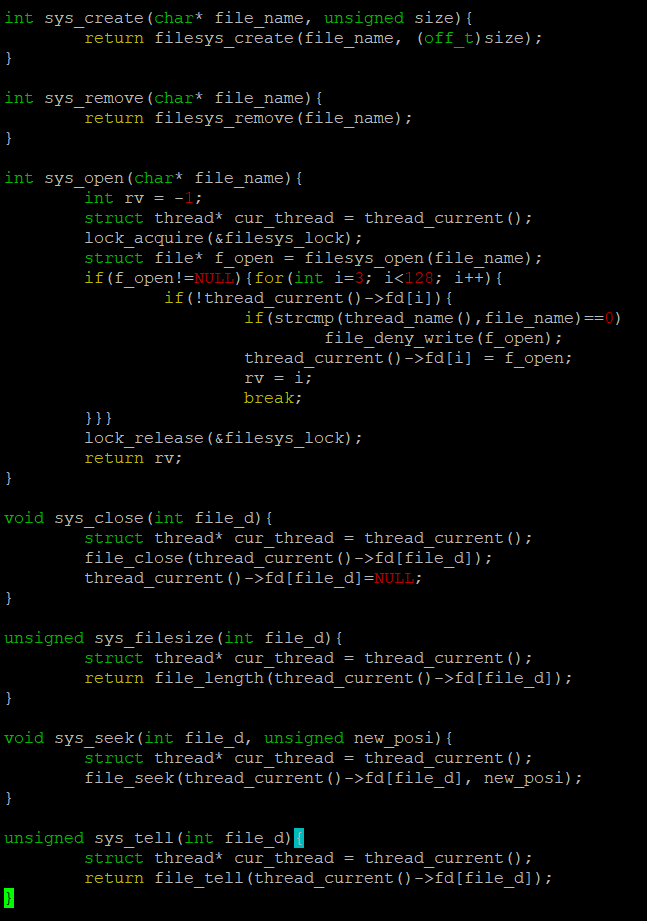


thread.h에서 thread구조체 내에 file descriptor배열과 file process의 execution상태를 확인하는 set\_wait변수와 부모를 가리키는 thread구조체 포인터, file system에서 사용될 filesys semaphore, 그리고 현재 보고 있는 file을 저장하는 file구조체 포인터 변수들을 thread구조체 내에서 선언한다.



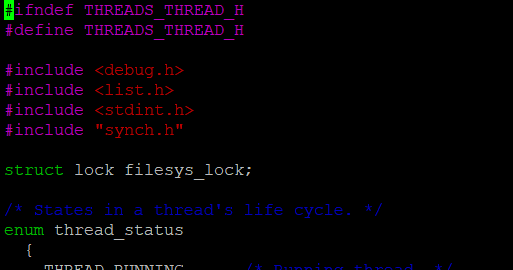
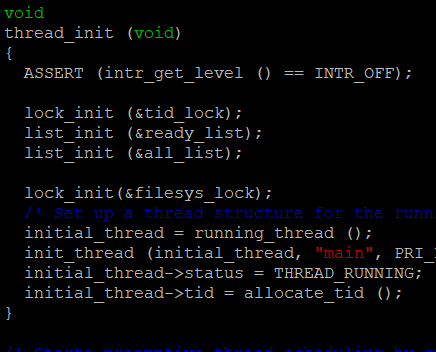
init\_thread함수 내에서 위에서 선언한 변수들을 초기화 해준다. 특히 file descriptor 배열의 원소들을 모두 NULL로 초기화한다.

2.

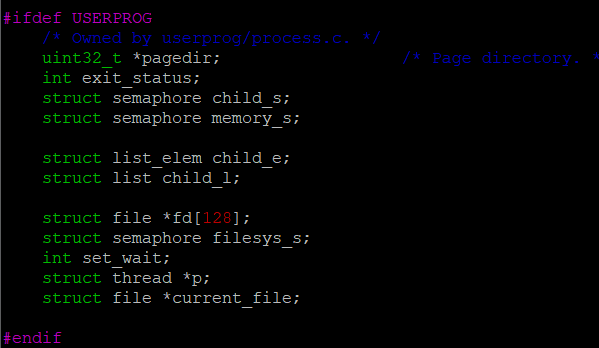


각 system call을 위해 syscall.c에서 작성한 함수이다. (선언은 syscall.h에서 해준다) sys\_create은 file명과 size를 입력 받아서 filesys\_create함수를 호출한다. sys\_remove는 file명을 입력 받아서 filesys\_remove함수를 호출한다. sys\_open은 파일명을 입력 받아서 filesys\_open함수를 호출한다. 이때, lock\_acquire과 lock\_release함수를 사용해서 critical section을 보호한다. close는 file descriptor을 입력 받아 filesys\_close함수를 호출하여 그 파일을 닫아주고, 할당 받았던 file descriptor값을 다시 NULL로 하여 재사용되게 한다. sys\_filesize는 파일명을 입력 받아 해당 파일의 file\_length함수를 호출한 값을 반환한다. sys\_seek은 file descriptor와 위치를 입력 받아 file\_seek함수를 호출한다. sys\_tell은 file descriptor를 입력 받아 file\_tell함수를 호출한 값을 반환한다.

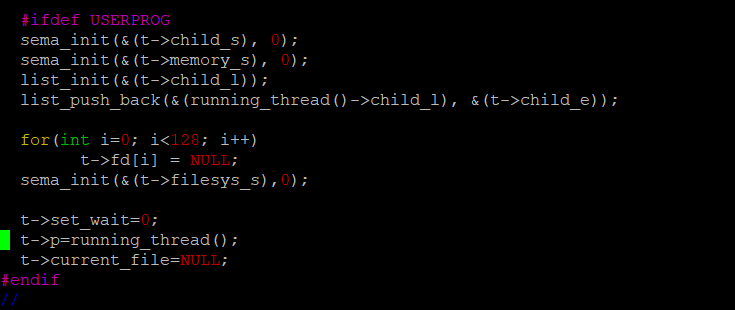
3.

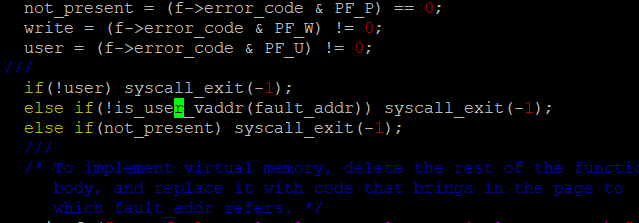
앞서 언급했던 lock을 통해 critical section을 보호하기 위해 filesys\_lock이라는 이름의 lock 구조체를 선언하고, thread\_init함수에서 초기화를 한다.



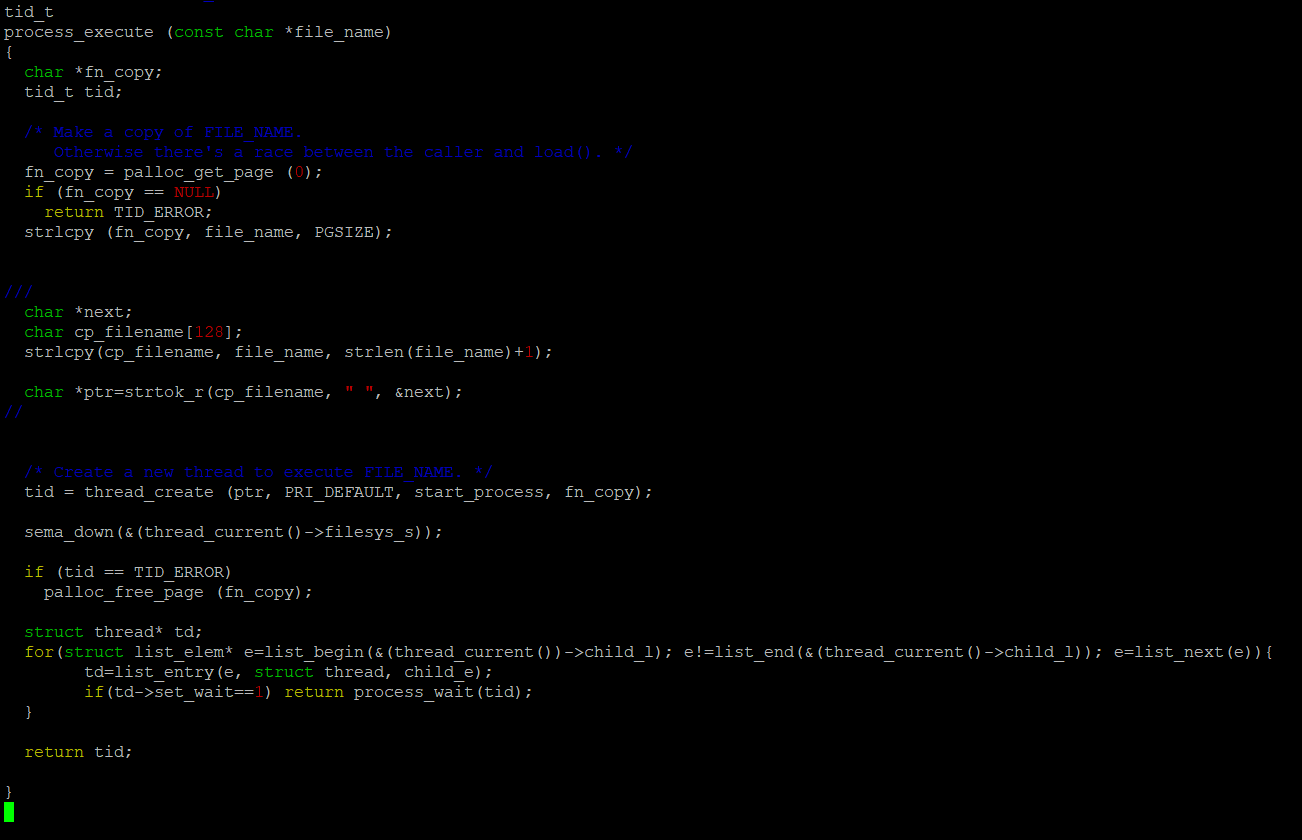
thread.h에서 thread구조체 내에 file descriptor배열과 file process의 execution상태를 확인하는 set\_wait변수와 부모를 가리키는 thread구조체 포인터, file system에서 사용될 filesys semaphore, 그리고 현재 보고 있는 file을 저장하는 file구조체 포인터 변수들을 thread구조체 내에서 선언한다.

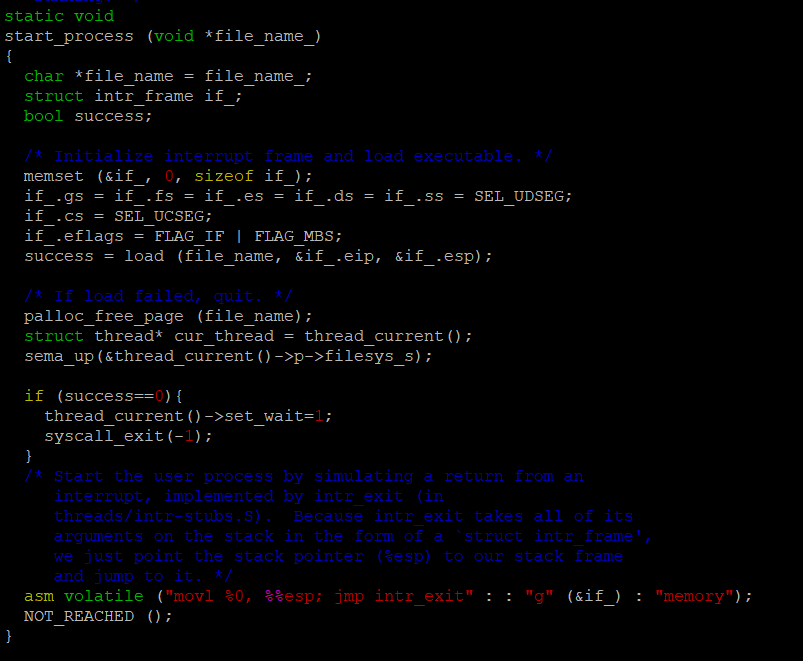


init\_thread함수 내에서 위에서 선언한 변수들을 초기화 해준다.



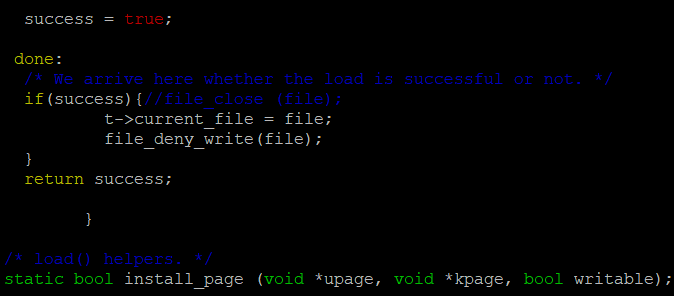
not-present page 관련 test를 통과하기 위해 exception.c에서 page\_fault 함수에 not\_present가 true면 exit(-1)을 호출하는 코드를 추가해준다.





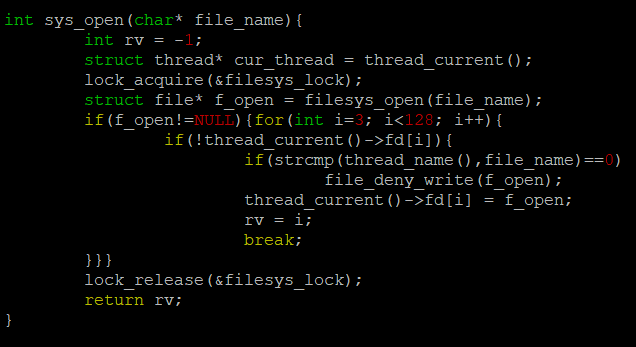
process.c 파일의 process\_excecute함수와 start\_process를 위와 같이 수정한다.

이는 filesys\_s라는 이름의 semaphore와 sema\_down함수, sema\_up함수를 사용해서 process가 load되기 전에 부모 process가 끝날 때나 현재 process가 작업을 문제 없이 끝냈을 때에만 memory 정리가 이루어지도록 하였다. 뿐만 아니라 start\_process함수에서 success==0이면 set\_wait을 1로 set하고, process\_execute함수에 현재 process의 set\_wait값을 확인함으로써 wait을 할지를 결정하는 코드도 추가해서 현재 process가 비정상적으로 종료될 때의 처리를 해주었다.

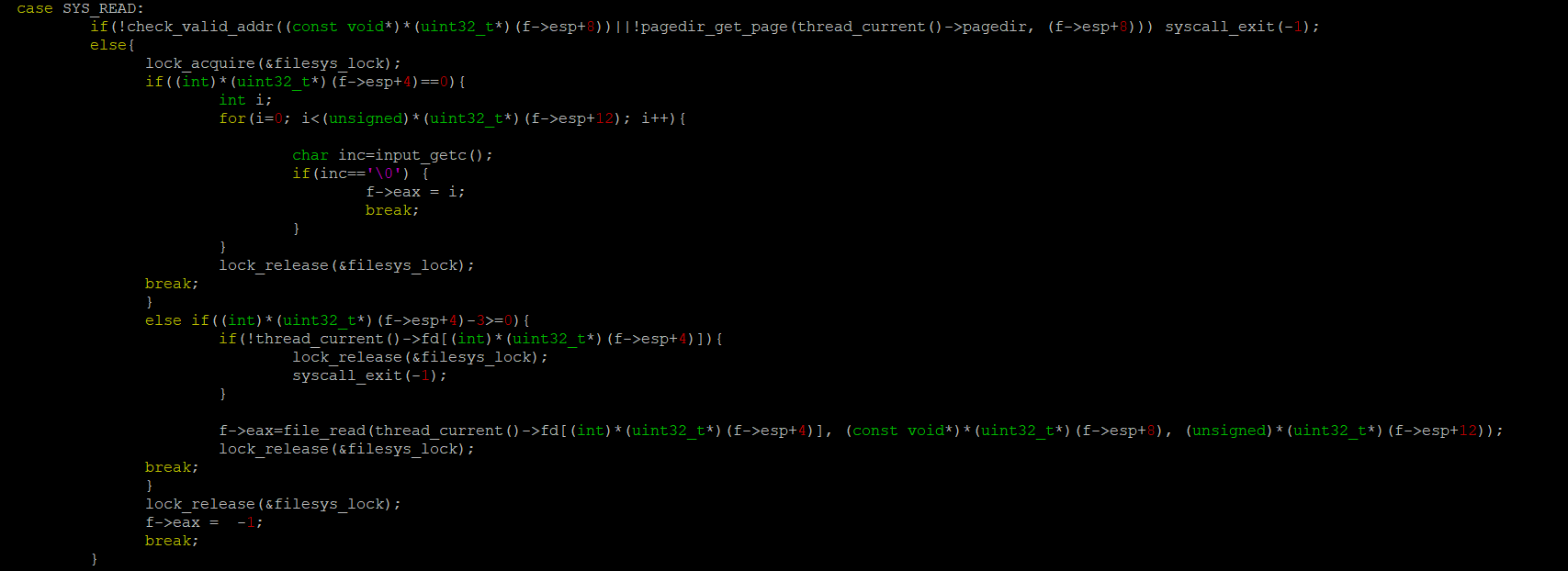


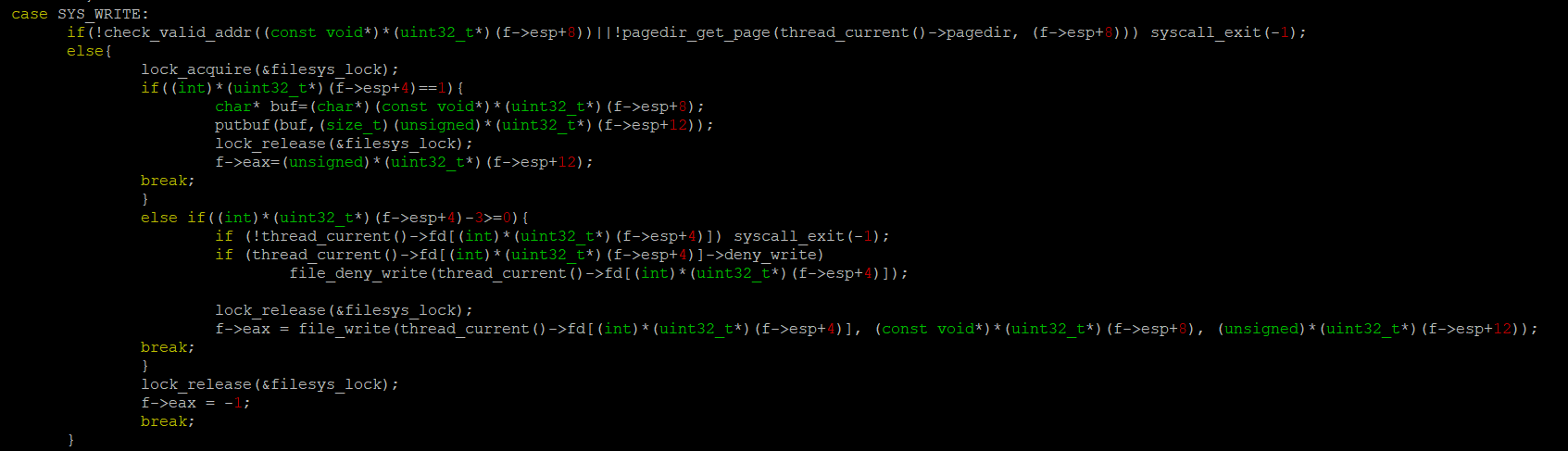
load함수에서는 success가 true인 경우 file\_deny\_write함수를 호출하여 write를 막는 코드를 추가해준다.

이제 syscall.c의 함수들에서 lock을 통해 동기화를 구현하면 된다.

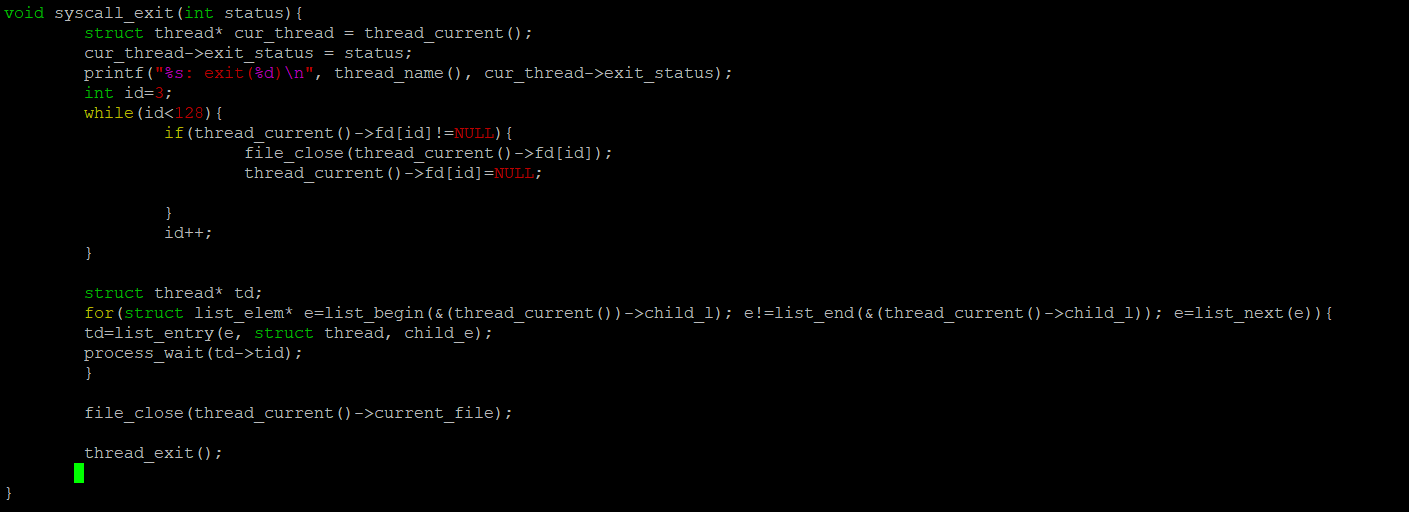


먼저 sys\_open 에서는 맨 처음과 맨 끝 부분에서 각각 lock\_acquire와 lock\_release를 호출함으로써 다른 process의 critical section 진입을 막았다.

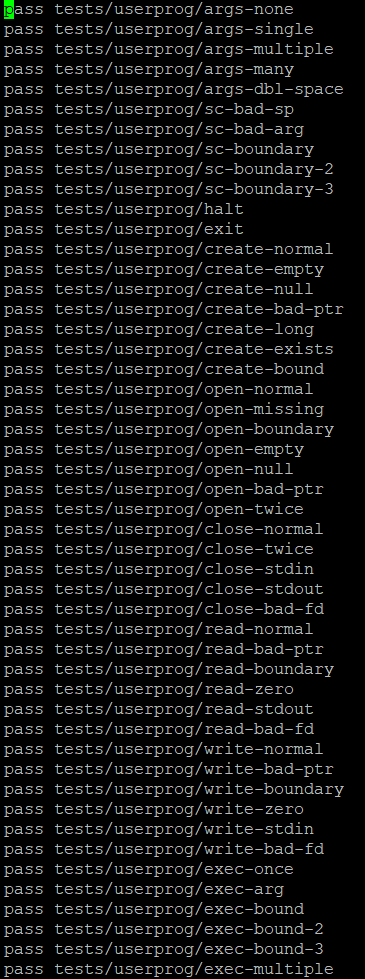




read와 write system call 부분에서도 역시 맨 처음과 맨 끝 부분에서 각각 lock\_acquire와 lock\_release를 호출함으로써 다른 process의 critical section 진입을 막았다. 이때 read의 경우 file descriptor가 0이면 standard input이므로 input\_getc함수를 사용해서 읽어주고, write의 경우 file descriptor가 1이면 standard output이므로 putbuf함수를 사용해서 쓰기를 실행했다. 또한 read의 경우 file descriptor가 3이상이면 read를 수행하고, write의 경우 file descriptor가 3이상이면 write를 수행하게끔 해주었다.



마지막으로 syscall\_exit함수도 수정해주었다. 부모 process가 자식 process를 기다리게 하기 위해서 for문을 통해 자식들을 기다리게끔 process\_wait함수를 불렀고, 마지막 부분에서는 file\_close로 open되어 있던 파일을 닫은 뒤, thread\_exit()함수를 호출하게 작성하였다.

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