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

담당 교수 / 분반 :김영채 / 2

이름 / 학번 : 김현우 / 20171628

개발 기간 : 10/3 ~ 10/12

1. **개발 목표**

file system관련 함수(create, remove, open, close, filesize, read, write, seek, tell)를 구현한다. 또한 critical section에 대한 mutual exclusion을 고려하여 synchronization이 적절히 잘 되도록한다. 그리고 file\_system function을 사용하여 denying writes to executable files를 구현한다. 마지막으로 프로세스 종료 이후 메모리 정리를 통해 메모리 누수를 막는다.

1. **개발 범위 및 내용**
   1. **개발 범위**
      1. user program에서 파일 시스템을 사용할 수 있도록 system call에 file system function들을 추가한다.
      2. 파일 시스템을 읽고 쓰는데 도움이 되는 시스템 콜이다.
      3. 이용중인 파일을 쓰거나 읽는 것을 방지한다.
   2. **개발 내용**

1. file descriptor 에는 file.c에 들어있는 file구조체를 활용했다. Thread를 생성할 때 각각의 스레드는 file descriptor 를 저장해야하는데, file descriptor 란 어떤 파일을 이용하는지에 대한 데이터의 모임이므로, file struct를 크기가 128인 배열로 만들어서 주소를 저장했다.

2. 구현한 system call과 설명

Bool create(): 파일의 주소, 크기를 받아서 파일을 생성한다. 성공하면 true 반환 실패하면 false 반환

Bool remove(): 파일의 주소를 받은 다음 파일을 삭제하고 결과를 반환. 반환값은 create 와 동일

Int open(): 이전의 open 함수와 달라진 점은 바로 파일을 열기전에 filesys\_lock함수로 file\_lock을 걸어줘서 다른 프로세스가 파일에 접근하지 못하게 만드는 점이다.

Int filesize(): 파일 디스크립터를 받아서 해당 파일의 파일 크기를 반환한다.

Int read(): file descriptor 와 읽을 공간인 buffer, 읽을 파일의 크기인 size, 를 인자로 넘겨 주고 파일에 쓰는 함수다. Fd 값이 0이면 키보드 입력이므로 키보드 입력을 읽고 아니 라 면 일반 파일을 읽기 시작한다.

Int write() : read와 동일한 파라미터를 받고 read 와 차이라면 파일을 쓸 때 파일 디스크 립터에 deny\_write가 참이면 현재 파일을 쓰지 못하게 만든다.

Void seek(): file descriptor 를 넘겨 받아서 현재 파일에서 읽는 위치를 이동시킨다..

Unsinged tell() : file descriptor 를 넘겨 받아서 현재 파일에서 읽는 위치를 판단한다.

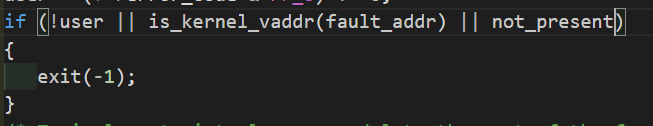
3. lock, semaphore 활용법

자식 스레드가 메모리에 적재 되기 전에 부모 프로세스가 종료가 되면 안되므로 자식 프 로세스가 load를 마치고 나서야 sema\_down을 통해서 부모 프로세스가 진행을 할 수 있 도록 한다. File system에서는 어떤 프로세스가 해당 파일을 읽을 때에는 다른 프로세스는 파일에 접근하지 못하도록 lock을 건다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

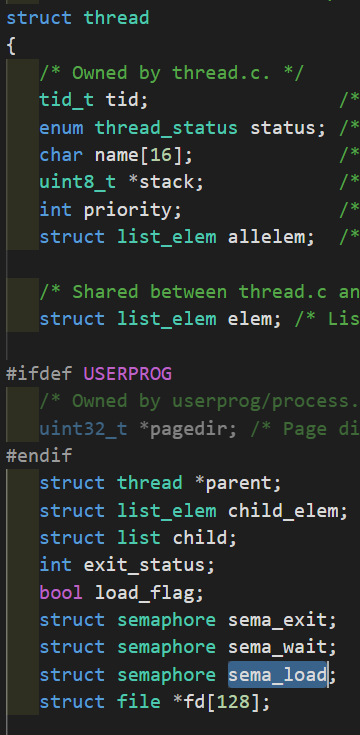
|  |  |
| --- | --- |
| 기간 | 내용 |
| 1주차 (10/3 ~ 10/10) | * 매뉴얼 분석 * 자료구조 및 알고리즘 설계 * file system, denying writes to executable files 구현 |
| 2주차 (10/11 ~ 10/14) | * 보고서 작성 |

* 1. **개발 방법**



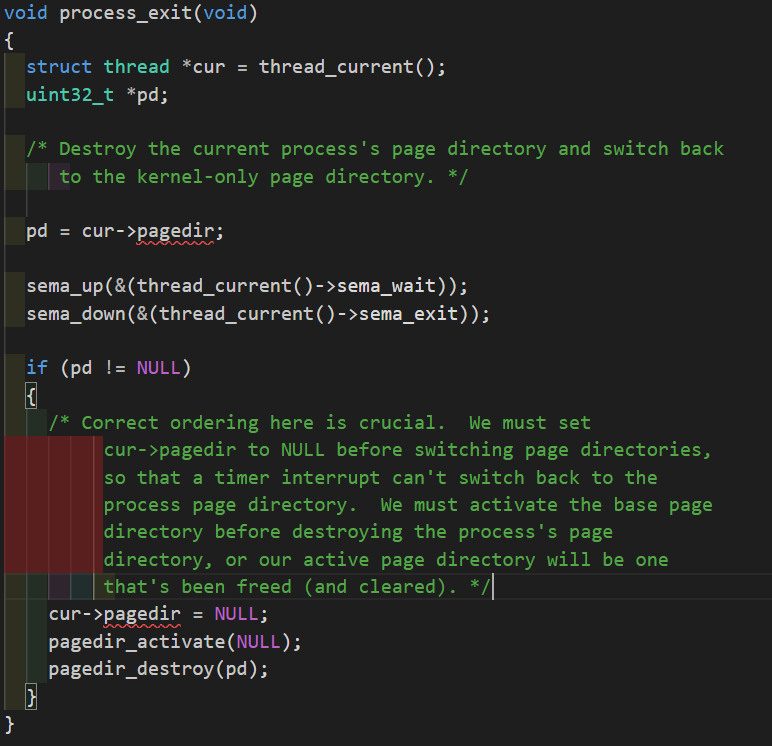
Stdin/stdout 의 경우에서 NULL이 주소로 들어오면 오류가 발생하므로 NULL이 들어온 다 면 exit(-1)을 실행시켜 프로세스를 종료시킨다.

**Thread.h에서 thread**



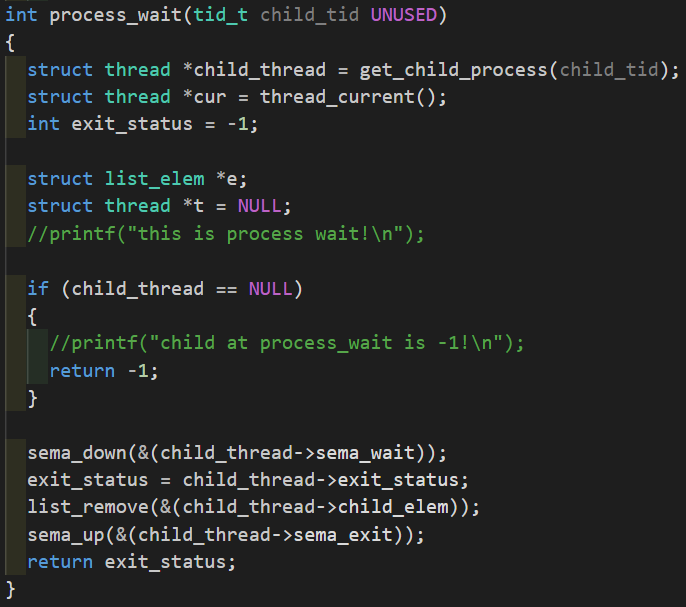
load에서 sema\_phore를 위해서 load\_semaphore 를 새로 만들고 파일 디스크립터 역할 을 할 file 포인터의 배열을 새로 생성한다.

**Process\_exit()**



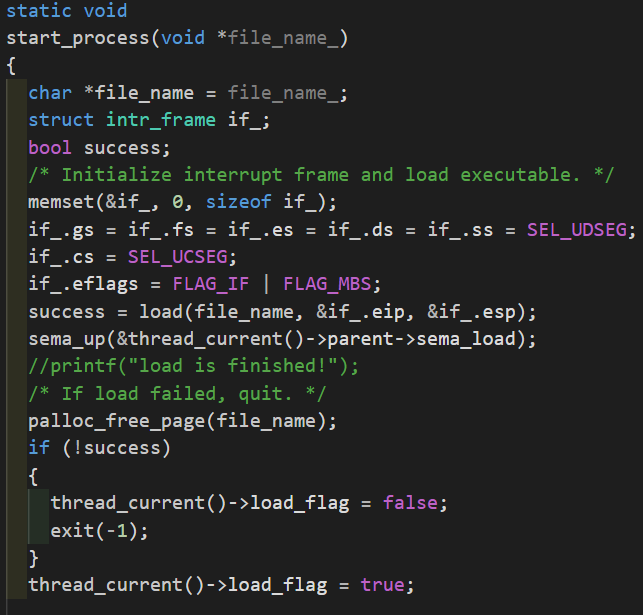
원래는 thread\_exit에 있던 세마포어를 이곳으로 이동시키니 오류가 없어져서 여기로 이 동하였다.

**Process\_wait**



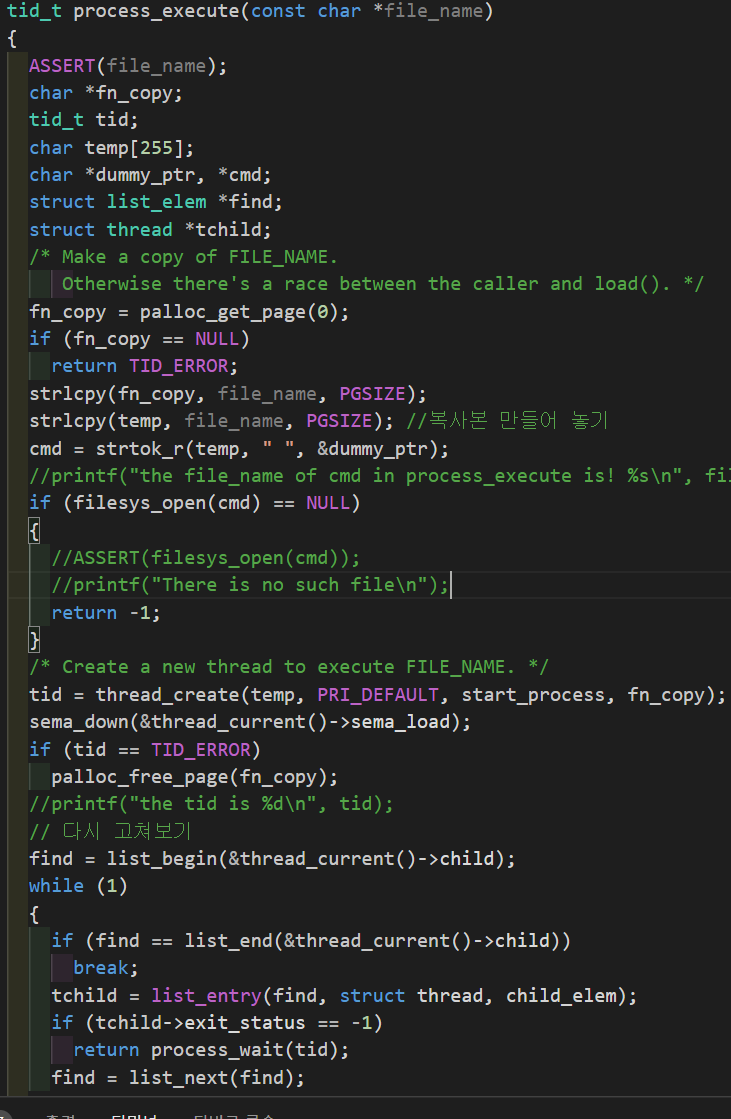
기다리는 동안 자식 스레드가 있다면 해당 스레드의 자식들을 초기화 시키고 부모 스레 드에게 자신의 exit\_status를 넘겨준다.

**Start\_process**



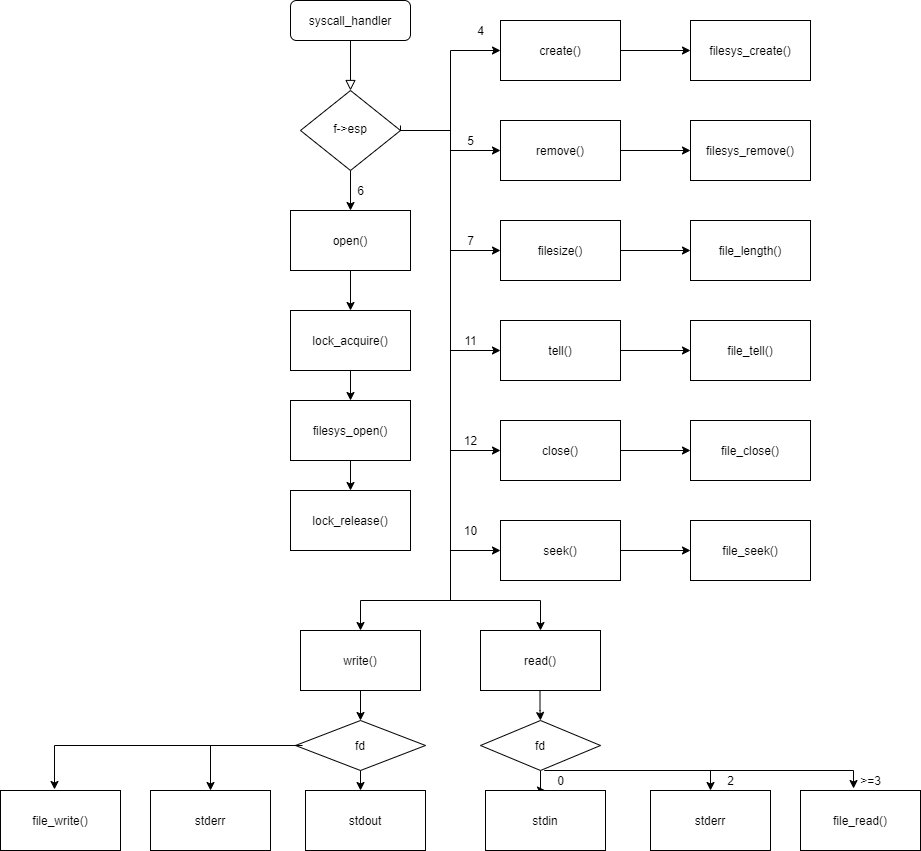
child thread의 load가 완료되면 parent thread가 진행될 수 있도록 start\_process에서 parent thread를 sema\_up 시켜줘야한다.

Process\_execute

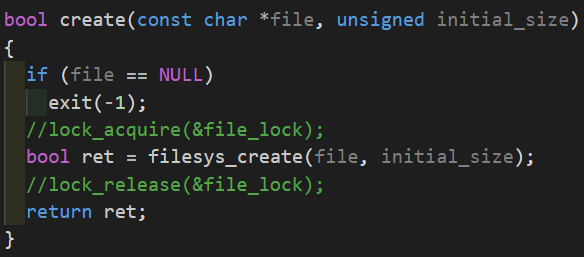


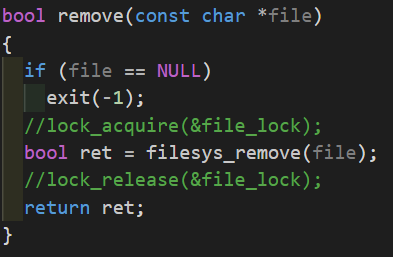
스레드를 만들고 sema\_down해서 프로세스를 중단하고 자식 thread가 load를 마칠 때까 지 기다린다. 이후에는 비정상적으로 종료된 자식 프로세스를 찾아서 종료 시킨다.

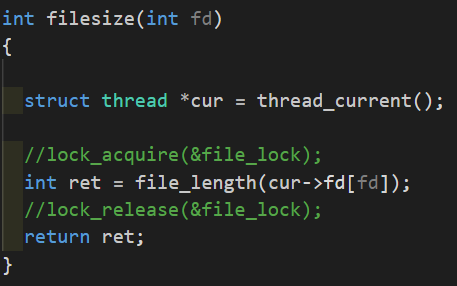
1. **연구 결과**
   1. **Flow Chart**

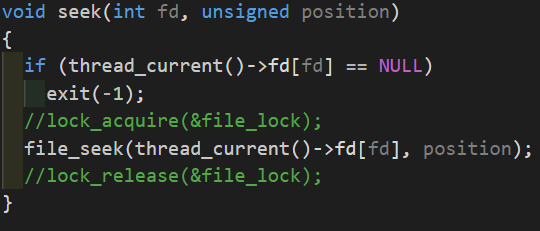


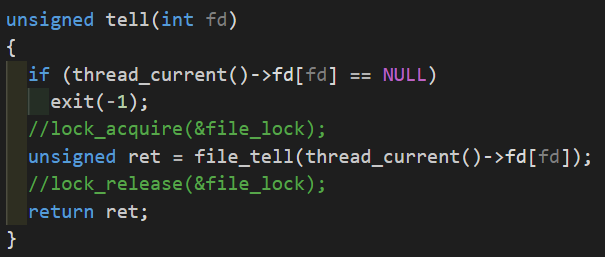
* 1. **제작 내용**
     1. user program에서 파일 시스템을 사용할 수 있도록 system call에 file system function들을 추가한다.





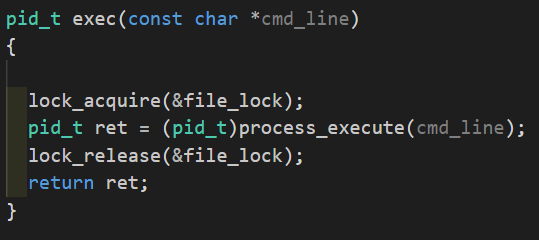


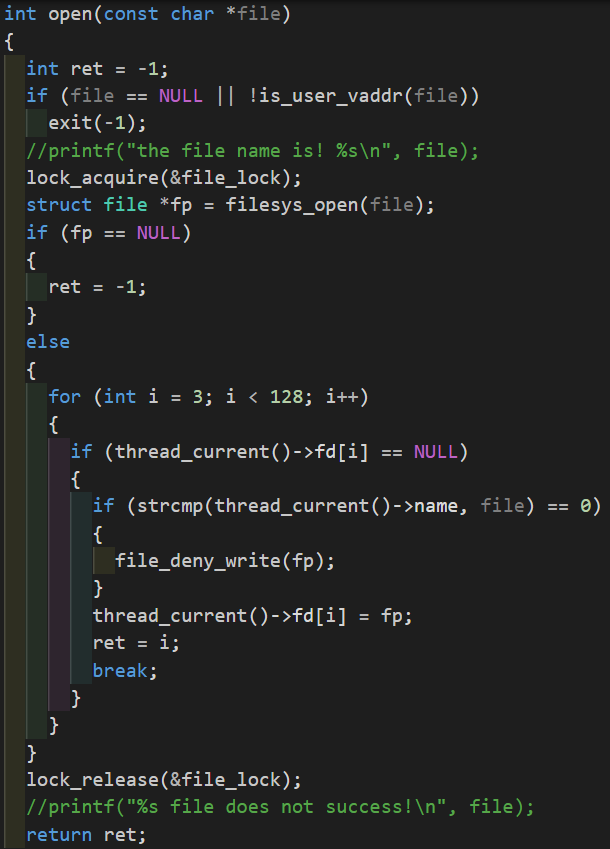




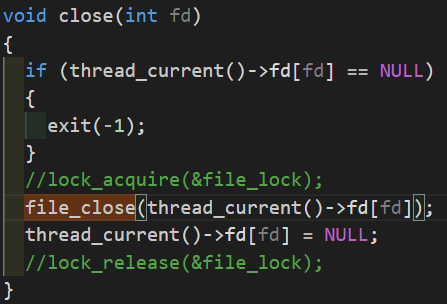
위의 시스템 콜은 filesys에 정의 되어있는 함수를 불러와서 해당 반환 값을 그대로 다시 반환 해주면 된다.

다음에 나오는 시스템 콜이 중요하다.

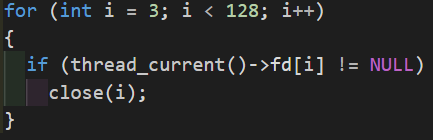




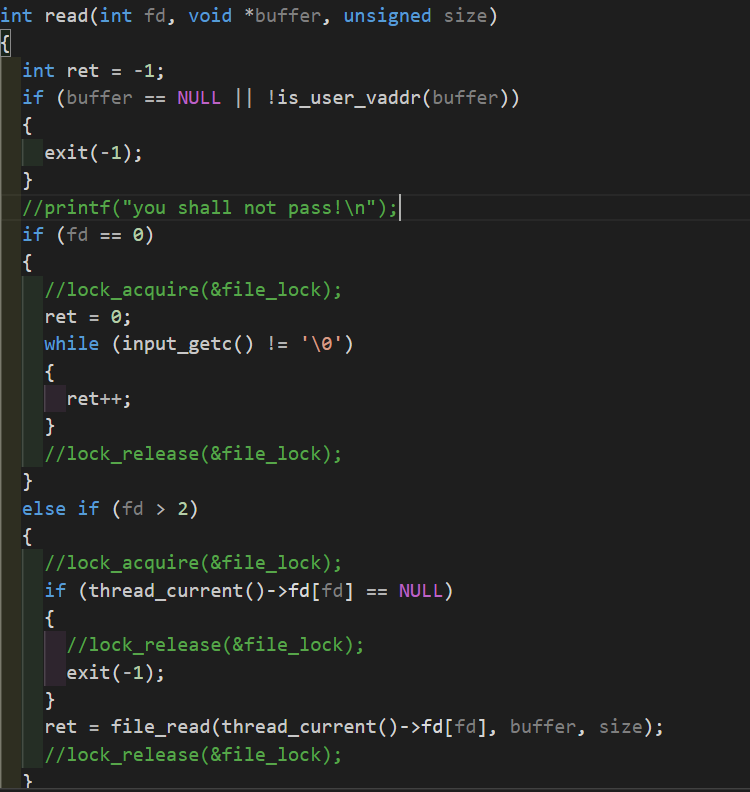
Filesys\_open 함수가 있으므로 앞뒤로 lock을 걸거나 해제해주는 함수를 배치한다. 또한 file descriptor를 탐색하면서 빈 file descriptor를 찾으면 일단은 해당 descriptor에 함수 주소를 넣어준다. 또한 실행하려는 파일이 현재 실행하는 파일과 이름이 일치한다면 file\_deny\_write를 통해 권한을 제한한다.



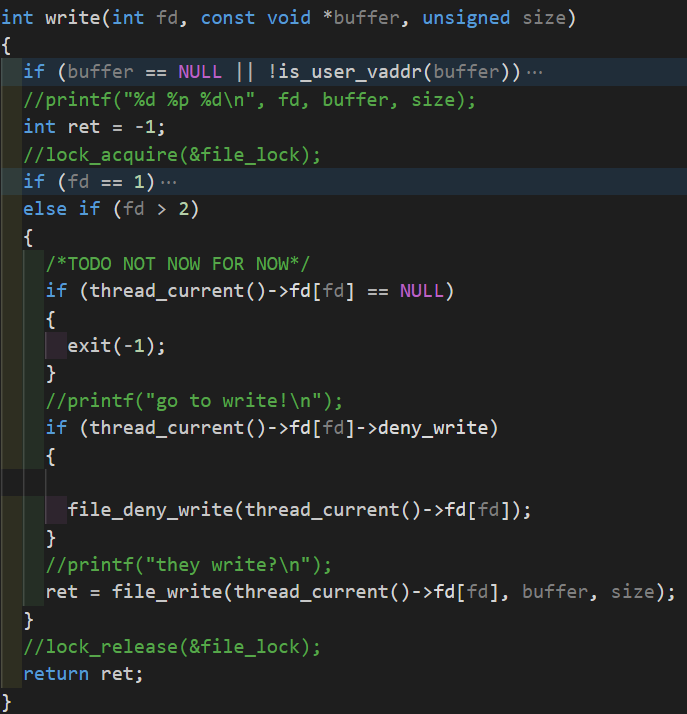
Close 함수에서는 단순히 file\_close를 부르고 끝이 아니라 꼭 닫은 파일 디스크립터에 NULL값을 넣어줘야 한다. 그렇지 않으면 exit 함수에서



이렇게 루프를 거치면서 NULL이 아닌 값에 close를 하게 되는데, 이미 닫아서 해당 파일이 존재하지 않아서 kernel panic이 호출이 된다. 그렇기 때문에 파일을 닫고 나면 꼭 NULL을 처리해줘야 하는 것이다.

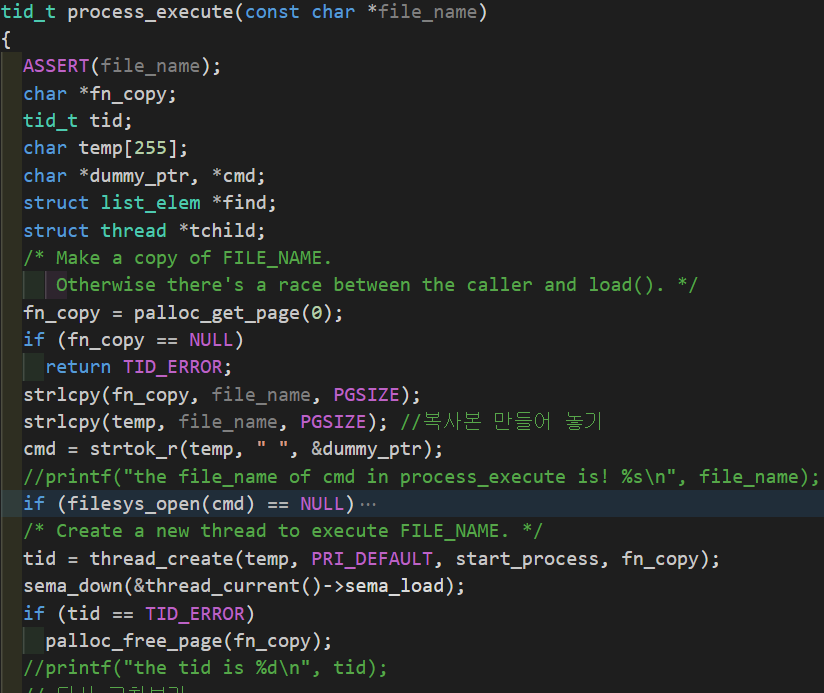


Read 함수는 지난 번과 달라진 점은 file descriptor 가 2 이상일 때의 경우를 추가하는 것인데, 만약 fd 값이 2 이상이라면 stdin stdout이 아닌 특정한 파일을 가르키는 것이므로 가르키는 주소가 없다면 exit(-1)을 호출한다. 만약 아니라면 file\_read 함수를 호출해서 반환 값을 받아온 이후에 반환한다.



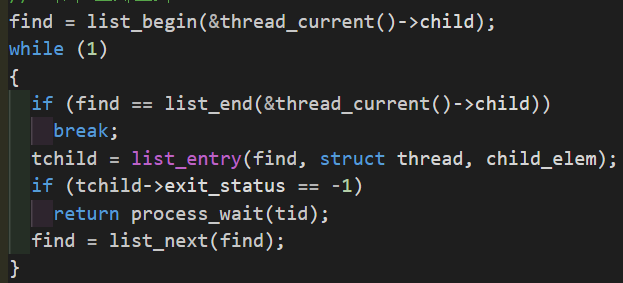
Write 함수도 fd가 2인 경우를 추가하였다. Read 함수와 다른점은 만약에 실행하려는 파일이 현재 실행하는 파일과 이름이 일치한다면 file\_deny\_write를 통해 권한을 제한한다. 그리고 read함수와 동일하게 file\_write 함수의 값을 반환해준다.

다음은 process.c 에 가서 파일 동기화를 해줘야 한다.

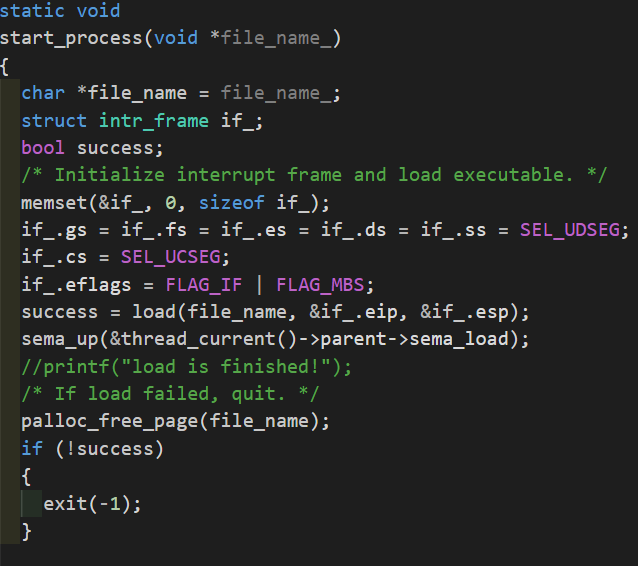


Process\_execute 가 thread\_create 로 thread를 생성하고 나서 문제가 생기는데 만약에 자식 프로세스가 로드가 실패한 상태라면 자식 프로세스는 나중에 실행은 되지 않을 것이다. 하지만 메모리에 load를 하지 않았어도 자원을 쓰게 되는데, 이게 누적이 되면 결국에는 메모리 누수가 발생한다. 이를 방지하기 위해 일단 thread\_create를 실행한 이후에 sema\_down을 해서 child\_process가 load 될 때까지 기다린다. 이렇게 child\_process가 load가 다 되어서 start\_process 에서 load가 완료되면, sema\_up을 통해서 다시 process\_execute를 진행한다.

이후에는 아래 코드대로 루프를 돌아 exit\_status 가 -1인 프로세스들을 종료시켜준다.



Process\_load 함수



위에서 설명했듯이 load함수 이후에 sema\_up을 실행해서 load가 완료되었음을 알린다.

분명히 lock 을 잘했다고 생각했는데, write-bad-ptr케이스 에서 지속적으로 lock함수와 관련해서 오류가 생겼다. 처음에는 write 와 read함수에는 lock을 무조건 해줘야 한다고 생각했는데, 코드들을 뜯어보니까 read와 write를 하기 전에는 반드시 open함수를 호출하는데 이것 때문에 lock관련해서 오류가 생기는 것이 아닐까 생각해서 write와 read함수에 있는 lock함수를 모두 제거하여 문제를 해결했다.

* 1. **시험 및 평가 내용**
* 