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

담당 교수 / 분반 : 문의현 / 2분반

이름 / 학번 : 지용환 / 20171300

개발 기간 : 11/5 ~ 11/13

1. **개발 목표**

프로젝트 1에서 구현한 시스템 콜에 추가하여 파일 시스템과 관련된 시스템콜을 구현하는 것이 목표이다. 관련된 시스템 콜은 create, remove, open 등등 이고 이때 critical section문제를 해결하기 위해 한 프로세스만 critical section에 들어갈수 있도록 동기화 기능을 구현하는 것이 동반되어야 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
2. File Descriptor

파일을 연 이후 그 파일에 접근할 때 file descriptor를 사용해야만 하기 때문에 구현이 필수이다. 프로세스들은 file descriptor를 이용해 thread별로 파일에 접근하며, 파일을 닫거나, 쓰거나 읽는 작업을 수행할수 있다.

1. (이번 프로젝트에서 구현해야 하는) System Calls

이전 프로젝트와 마찬가지로 system call handler를 통해 명령어를 처리할수 있으며 이번에 구현해야하는 명령은 파일 시스템에 관련된 것으로 파일의 생성, 제거, 열기, 사이즈 조회, offset조회, 읽기,쓰기, 닫기 이다.

3. Synchronization in Filesystem

read/write할 때 동시에 파일을 읽거나 쓰면 critical section의 문제가 생겨 데이터의 동기화에 문제가 생기기 때문에 이를 방지해주어야 하는데, 이를 위해 구현하는 것이 synchronization이다. 이를 구현하면 하나의 프로세스가 wirte중일 때, 다른 프로세스는 read,wirte를 못하게 함으로써 구현할수 있다. 뿐만 아니라 저번 프로젝트에서 처럼 프로세스가 execute/exit할때도 필요하다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

구현하기 위해선 file.h에서 제공하는 struct file\*의 배열을 사용했는데, 배열이 만들기 쉽고 참조도 간편하기 때문에 특히, 피놑스 매뉴얼에서 fd의 max값은 128이기에 크기가 128인 배열을 생성했다. 각 프로세스 별로 file descriptor를 가지고 잇어야 하기에 thread.h에서 file\_d로 선언했다.

1. System Calls: 구현할 각 system call에 대해 간략히 서술 (하나의 system call 당 최대 3문장으로 간략히 설명; 3문장을 넘길 정도로 길게 작성하지 말 것)

create: 주어진 파일 이름과, size를 통해 filesys\_create를 호출하고 새로운 파일을 생성한다. 성공시 true, 실패시 false를 반환한다.

remove: 주어진 파일 이름에 대해 filesys\_remove를 호출해 그 파일을 삭제한다. 성공 여부를 (true/false)를 반환해준다.

open: 주어진 파일이름을 통해 해당 파일을 연다. 파일 여는 것에 성공하면 open 파일에 대한 file desciptor를 반환하고, 실패한다면 -1을 return 해준다. 만약 파일이 사용중이라면, 쓰기를 deny시켜준다.

filesize: 주어진 file descriptor에 해당하는 파일의 byte size를 return한다.

seek: 주어진 file descriptor에 해당하는 파일의 읽거나 쓸 다음 위치로 바꿔준다.

tell: 주어진 file descriptor에 해당하는 파일의 읽거나 쓸 다음 위치를 return한다.

close: 주어진 file descriptor에 해당하는 파일을 닫는다.

read: 주어진 file descriptor로 열리는 file를 size만큼 buffer로 읽는다. read성공시엔 읽은 size를 반환, 실패시엔 -1을 반환한다. fd=0은 stdin을 의미한다.

write: buffer에서 size만큼을 file descriptor에 해당하는 파일에 쓴다. wirte성공시엔 쓴 byte size를 반환, 실패시엔 -1을 반환한다. fd=1은 stdout을 의미한다.

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

read, write를 동시에 한다면 위에 설명했듯이 문제가 생기기 때문에 이를 방지하기 위해 미리 선언된 lock 구조체를 사용했다.(file에 관련된 것이기에 file\_lock으로 재선언함) 작업을 하기전에 lock을 acquire해주고, 작업이 끝났을 때 lock을 release 해줌으로서, 하나의 작업을 하는 중에는 lock을 얻지 못해 동시에 여러 작업이 수행되는 것을 막아 synchrnizaion을 구현할수 있다. 물론 semaphore를 이용해서도 할수 있었지만(그냥 작업전에 sema\_down, 작업이 끝나면 sema\_up을 시켜주면 결과는 동일할 것이다.) 이미 lock을 사용했기에 프로세스간의 동기화처리에서 semaphore를 사용했다. ope,cloase등의 함수가 불려 process간 파일의 execution/load의 순서를 정해야 하기 때문에 프로세스의 synchronizaion이 필요했는데 process\_execution에서 sema\_down, 그리고 start\_process에서 sema\_up을 시켜 프로세스의 동기화를 처리하였다. 당연히 프로세스마다 필요한 것이기에 thread.h에서 s3이란 semaphore를 새로 선언해 주었다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

11/5~11/9 file descriptor, system call 구현

11/9~11/12 synchronization 구현

11/13 보고서 작성

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

thread.c, thread.h, syscall.c, syscall.h, excipton.c, process.c

* + 수정하거나 추가해야 하는 자료구조
  + 수정하거나 추가해야 하는 함수

thread.h에서는 thread구조체를 변경해 주어야 하는데, file decriptor역할을 수행하기 위한 크기 128 file 구조체 배열을 선언해준다. 또한 프로세스간 동기화(child가 실행중인지 관리)를 위한 semaphore s3. 을 선언해 준다. 또한 현재 실행중 file을 가리키는 file구조체, 부모 thread 그리고 자식이 정상적으로 종료된지 확인 가능한 flag변수를 선언해주면 편하다.

thread.c에서는 위 thread에서 선언한 새로운 변수들을 초기화 해주는 코드를 작성해야하는데, flag=0., file desciptor배열과 현재 실행중인 파일 구조체는 모두 null로, s3은 sema\_init으로, 부모 thread는 running thread로 초기화 해준다.

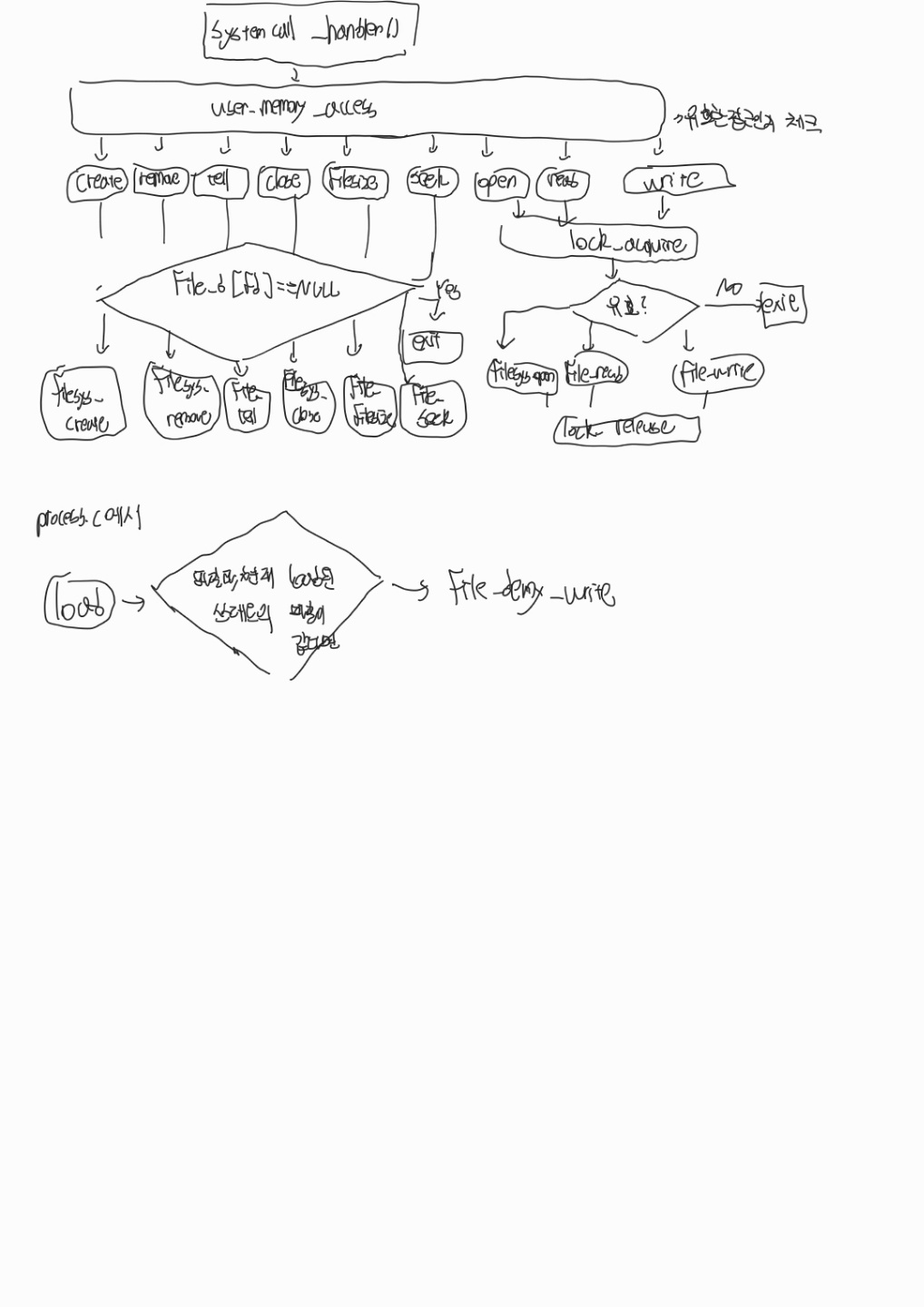
syscall.c 에선 구현하기 위한 모든 system call을 구현해주는 코드를 작성해주면 된다. 이전 프로젝트와 마찬가지로 handler에 각 system call을 등록하여 user\_memory\_access함수를 통해 옳은 접근인지 확인한 후, 각 경우에 맞는 함수를 실행하고, return값은 eax에 저장해준다., read, wrtie를 제외한 모든 새로운 시스템 콜은 미리 정의된 filesys\_ 함수를 통해 구현할수 있고, 단순히 인덱싱하는 fd를 통해 file descriptor에 접근, 그리고 각 상황에 맞는 작업을 취해주면 된다. read, write의 경우에는 syncrhonization문제를 위해 file\_lock을 acquire, release 하였고, 오류 상황이 아닌 경우 미리 정의된 file\_함수를 통해 구현하였다.

exception.c 구현을 하다가 안것인데, kernel panic이 뜨는 경우를 방지하기 위해 page\_fault에서 조건을 강화해주어야 한다. not\_present가 0인 경우도 exit 시켜주는 코드를 추가한다.

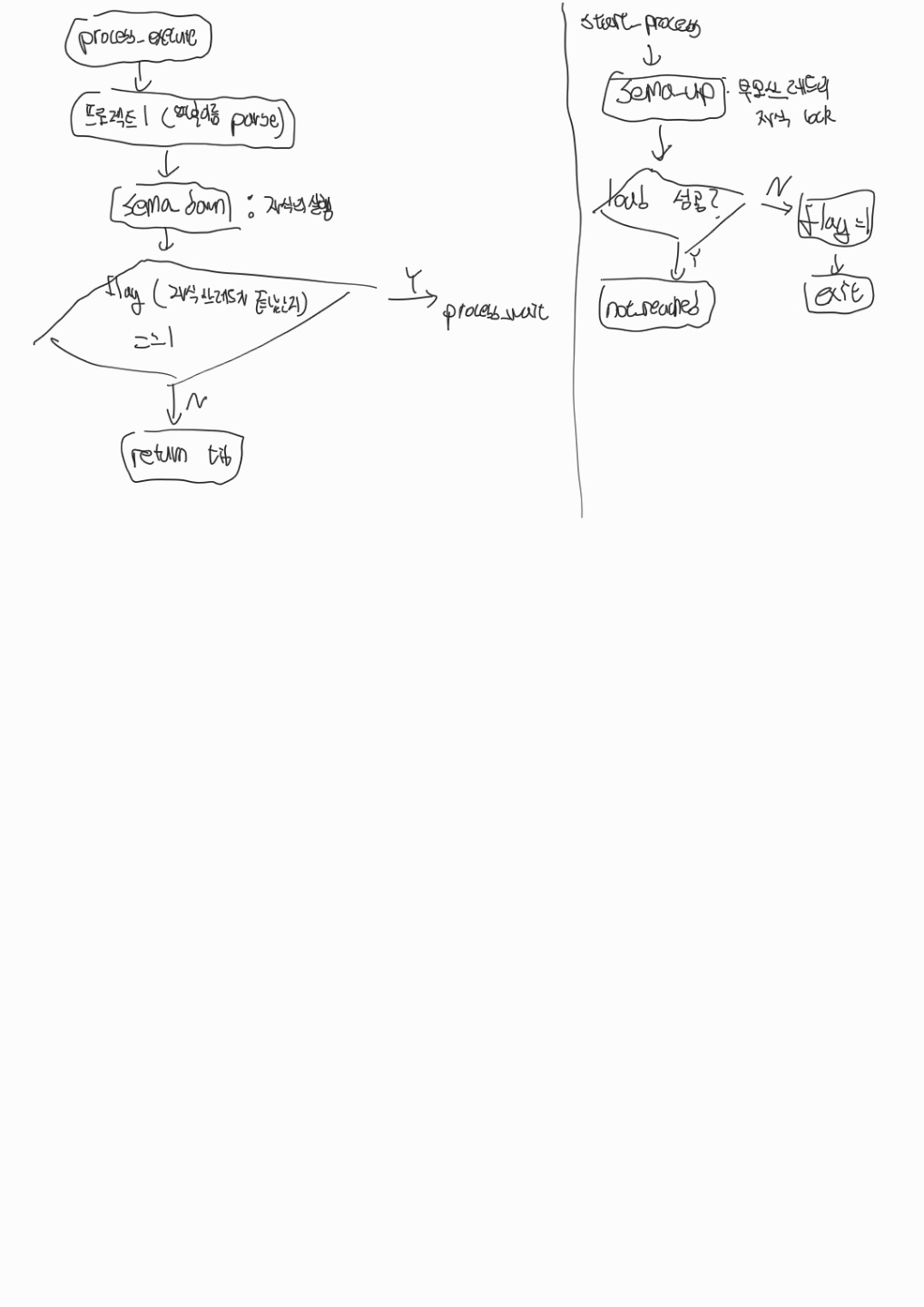
process.c 에선 프로세스 동기화를 구현해주면 된다. semaphore를 이용할 것이기 때문에, process\_exectiopn과 process)\_start에서 sema\_up과 sema\_dowm을 통해 parent가 child를 기다리게 해준다. 이때 비정상적으로 종료된 child 처리를 위해 flag변수도 활용한다. 또한 정상적으로 load가 되었을 때 해당 file을 current\_file로 저장하고, file\_deny\_wirte하여 실행중인 thread에 write하는 경우를 방지해주어야 한다. 정상적 파일 종료를 위해 syscall의 exit에서 file\_close를 이용해 현재 file을 종료시켜준다면 파일의 종료도 정상적으로 수행될수 있다.

1. **연구 결과**
   1. **Flow Chart**
2. **file descriptor**텍스트이(가) 표시된 사진

   자동 생성된 설명
3. **system call**



1. **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. **File descriptor**

**thread/thread.h**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

file descriptor를 위해서 선언된 변수는 file\_d[128]하나 뿐이다. 나머지는 나중에 설명하겠다.

thread/thread.c



이전 프로젝트에서 thread의 초깃값을 설정하는 init\_thread에서 쓰레드의 file\_d를 모두 NULL로 초기화 시켜준다.

* + 1. System call

userprog/syscall.h

이전 프로젝트에서처럼 이미 lib/user/syscall.h에 정의되어 있는 그대로 추가해 주면된다.(read, write 동일

텍스트이(가) 표시된 사진

자동 생성된 설명)

userprog/syscall.c

이전 프로젝트와 동일하게, system call handler에 추가적으로 구현하면 된다. 물론 user 메모리 영역인지 확인하는 user\_memory\_access함수도 사용하는것도 동일하고, return값은 eax에 넣어주는것도 동일하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 create 함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

파일이 null이라면 exit(-1)로 종료시키게 하였고, 그외엔 filesys\_create로 파일을 생성하도록 하였다. 이때 filesys\_create는 파일이름(file)과 initial size를 받아 그 size만큼의 파일을 생성하는 함수로, 받은 file이름이 이미 존재하는 것과 같은 이유로 생성 실패시, false, 생성 성공시 true를 반환한다.

다음은 remove함수이다.

텍스트, 오렌지, 어두운이(가) 표시된 사진

자동 생성된 설명

create와 동일하게 null이 아니라면 filesys\_remove를 동해 파일을 삭제하도록 하였다. 이때 filesys\_remove는 파일 이름을 받아 해당 파일을 삭제하는 것으로 받은 파일이름이 존재하지 않는 이유 등으로 삭제 실패시 false 성공시 true를 반환한다.

다음으로 open함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

받은 파일이 null이라면 exit로 종료시키고 그 외의 경우엔 open중에 접근이 불가능하도록 lock\_aquire를 시켜준다. filesys\_open으로 파일을 열어주었다. 이때 filesys\_open은 해당 파일을 열어 성공시에 file의 포인터를 반환, 실패시엔 null을 반환시켜주는 함수이다. 즉, fp가 null은 open에 실패했다는 뜻이므로 이때도 exit를 시켜준다. 그 외엔 현재 trhead의 file descriptor의 빈자리를 for문으로 찾아, null(즉 빈자리)인 부분에 파일 포인터를 저장한 이후, release시켜준다.

다음으로 filesize이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

fd를 받아 fd에 해당하는 파일에 접근하고, 없다면, 즉 file desciptor에서 null이라면 exit 그 외엔 file\_length의 결과를 반환해 준다. 이때 file\_length는 파일 포인터의 파일 사이즈를 반환해주는 함수이다.

다음으로 seek이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

filesize와 유사하게, fd에 해당하는 파일을 찾아 없으면 exit 그외에는 file\_seek함수를 사용한다. 이때 file\_seek은 파일 포인터와 position을 받아 파일의 현재 위치를 시작점으부터 position만큼의 위치로 바꾸어 주는것으로써, 쉽게 file의 위치를 바꿔주는 함수이다.

다음으로 tell이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

fd를 받아 없으면 exit 있다면, 파일의 시작점부터 현재 위치를 byte offset로 반환해서 return하는 함수이느 file\_tell을 return시켜준다.

다음으로 close이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

fd를 받아 없는 파일이면 exit 그외에는 파일 포인터를 받고 해당 파일을 닫아주는 함수인 file\_close를 활용해 파일을 닫았고, 이후에 파일이 닫혔으니, file descriptor의 부분을 null로 바꿔준다.

다음으로 read함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이전 프로젝트에서 fd가 0이 아닌 경우도 추가한 것으로 이때 동시에 write read가 되는 상황을 방지 하기 위해 lock acquire를 사용해준다. fd>=3인 경우에 새롭게 추가한 코드로 만약 해당 fd의 파일이 없다면 exit, 그외엔 file\_read를 사용해 결과를 얻고 그 결과를 반환시켜준다. (당연히 모든 case종료전에 lock\_relase는 필수이다.) 이때 file\_read는 파일 포인터(file), buffer, size를 받아 파일의 내용을 현재 position으로부터 size만큼 읽어서 buffer에 저장시키고, 실제로 읽은 byte size를 반환해주는 함수이다.

다음으로 write함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

read와 거의 동일하다. lock\_acquire를 사용해 동시 접근을 막고, fd>=3이상인 경우에 해당 파일이 없다면 exit, 그외엔 file\_write를 이용한 후에 결과를 반환시켜준다. 이때 file\_wirte는 file포인터, buffer,size를 받아 buffer의 내용을 size만큼 읽어 fp의 현재 위치부터 쓰는 함수로, 실제로 쓴 byte size를 반환시켜준다.

테스트를 하는 중에 계속 kernel panic이 떠서 해당 원인에 대해서 찾아보니, exception.c에서 예외 조건을 좀더 강화해주면 된다는 글을 보았다.



이 코드를 page\_fault에 추가하였더니, kernel panic은 사라졌다.

또한 문제중에 bad\_read\_ptr만 자꾸 fail이떠서(bad-write는 성공했다..) 뭐가 원인인지 도통 모르겠어 그냥 read함수안에서 한번더 user\_memory access를 활용해 buffer를 한번더 검사했더니, bad\_read\_ptr을 pass하였다. 이유는 아직도 모르겠다..

텍스트이(가) 표시된 사진

자동 생성된 설명

* + 1. Synchronization.

먼저 file\_lock을 초기화 해주지 않고 사용해 자꾸, 실행조차 안되는 에러가 생겼었고, 이를 해결하기 위해 thread.c의 thread\_init에서 lock\_init을 사용해주면 해결된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 위의 system call의 open, write,read에서 사용한 lock\_acquire과 lock\_relase가 모두 synchrnization에 관련된 것이다. 다만 이미 설명했으니 생락하겠다.

thread.h

텍스트이(가) 표시된 사진

자동 생성된 설명

file\_d를 제외한 모든 변수(exit\_c는 이전 프로젝트)가 synchrnization에 관련된 것이다. 먼저 s3은 프로세스간 동기화, 즉 부모가 먼저죽는 상황을 방지하기 위해 선언한 변수이고, 자식이 제대로 종료되었는지 아는 변수인 flag, 그리고 parent변수도 프로세스간 동기화를 위한 변수이다. 또한 실행중인 파일에 write를 하지 못하도록 하기 위해 current\_file이란 file구조체도 선언해 주었다.

thread.c

텍스트이(가) 표시된 사진

자동 생성된 설명

thread.h에서 선언한 변수를 모두 초기화 해주면 된다. s3은 세마포어 변수를 초기화하는sema\_init으로그리고 parent는 현재 실행중인 스레드를 반환해주는 running\_thread, flag는 0, current file은 null로 초기화 해준다.

userprog/process.c

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

process\_execution부분에서 프로세스간 동기화를 위해 스레드가 create된 이후, 바로 sema\_down을 시켜주어 parent가 자식보다 먼저 죽는 경우를 방지한다. 이후, child스레드를 for문으로 검사하며 flag==1즉 후술할 start\_process에서 비정상적으로 load된 자식 프로세스를 process\_wait를 이용해 처리시켜준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

start\_process부분에선 정상적으로 load되었을떼 부모 process의 s3을 sema\_up시켜준 이후, execution함수에서 부모 프로세스를 재 실행하게 해주고, 로드에 실패한 경우 flag=1을 만든후, 이를 process\_execution에서 처리하게 만든다.

.

텍스트이(가) 표시된 사진

자동 생성된 설명

load함수에선 성공적으로 load를 했을 때, 현재 스레드의 current\_file에 load된 파일을 저장하고, file\_deny\_write를 호출해 해당 파일에 wrtie하는 것을 막아주었다. 이때 file\_deny\_write는 파일 포인터를 받아 해당 파일에 wirte하는 모든 연산을 막아주는 함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이는 syscall.c의 함수로서, 현재 스레드의 모든 파일을 close를 이용해 종료시키게 해주었고, 부모 스레드가 먼저 죽는 것을 막기 위해 procerss\_wait을 통해 기다리도록 하였다. child가 모두 종료된 이후에는 file\_close를 통해 현재 스레드의 current\_file을 close하고, thread\_exit를 통해 스레드를 종료시켜주었다. 이를 통해 실행중인 파일에 대한 write를 막아 줄수 있다.

* 1. **시험 및 평가 내용**
* make check 수행 결과를 캡처하여 첨부
* 텍스트이(가) 표시된 사진

  자동 생성된 설명