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

담당 교수 / 분반 : 문의현 교수님

이름 / 학번 : 노영현 / 20181264

개발 기간 : 11/9 ~ 11/14

1. **개발 목표**

Project 1에서 구현했던 system call들에 추가해 새로운 system call들을 추가한다. 이번에 새로 추가하는 system call은 create, remove, open, close, filesize, read, write, seek, tell을 추가한다. system call을 구현하는 과정에서 synchronization과 File descriptor 관련 구현이 필요하다. 이 또한 개발해 문제 없이 pintos system call을 구현하는 것이 이번 프로젝트의 목표이다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목을 구현해야 하는 이유, 혹은 구현 시 기대되는 결과를 간략히 서술

1. File Descriptor

File Descriptor란, 운영체제에서 파일의 입력/출력 리소스에 액세스하는 데 사용되는 표현이다. File Descriptor는 음이 아닌 정수값으로, 0번부터 2번까지는 고정되어 있다. 0번 부터 각각 표준 입력, 표준 출력, 표준 에러 출력을 의미한다. File Descriptor의 역할을 수행하는 1차원 배열을 개발 시에 직접 구현한다. 이를 사용해서 system call에서 프로세스가 file에 access할 수 있게 한다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

사용자가 system call 명령어를 입력하면 kernel 수준에서 프로그램을 실행시키게 해준다. Proj1에서와 마찬가지로 사용자가 user level에서 명령어를 입력하면 kernel level에서 프로그램을 돌아가게 한다. 이번 프로젝트에서는 create, remove, open, close, filesize, read, write, seek, tell system call을 구현해 user level에서 kernel level에서 위 명령어들을 실행할 수 있게 한다.

3. Synchronization in Filesystem

Proj2에서는 주로 file과 관련된 system call을 다룬다. system call을 사용해서 file에 접근할 때 현재 write하고 있는 file에 또 다른 프로세스가 같이 write 하려고 한다면 race condition이 발생한다. 이를 방지하기 위해서 이번 프로젝트에서 synchronization이 필요하다. 이를 critical section에서 lock등을 이용해 race condition이 발생하지 않도록 구현한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

File Descriptor을 구현하기 위해서 thread.h 파일의 struct thread 안에서 struct file \*fd[128]을 선언하였다. 128크기의 fd 1차원 배열이 File Descriptor을 나타낸다. 핀토스 메뉴얼에서 각 process가 open하는 file의 개수가 128개를 넘지 않는다고 명시하고 있으므로 크기는 128로 설정해주었다. 1차원 배열을 선언해서 각 인덱스에 접근할 수 있어 각 file descriptor를 다루기 용이하다.

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)

- sys\_create : 인자로 file 이름과 file의 초기 크기를 받는다. 새로 만드는 file의 이름을 첫 번째 인자로 설정하고, 초기 size를 두 번째 인자로 설정한다. 이에 성공했으면 true를, 실패했다면 false를 반환한다.

- sys\_remove : 인자로 file이름 하나를 받는다. 현재 가지고 있는 file들 중 인자와 같은 이름을 가지는 파일을 찾아 삭제한다. sys\_create와 마찬가지로 성공했으면 true를, 실패했다면 false를 반환한다.

- sys\_open : 인자로 file이름을 받는다. 인자로 받은 이름을 가지는 file을 찾아 open한다. 이후 file\_descriptor 배열에서 빈 곳에 file을 넣어준다. 파일 open에 성공했으면 true를, 실패한다면 false를 반환한다.

- sys\_close : 인자로 file이름을 받는다. 인자로 받은 이름을 가지는 file을 찾아 close한다. sys\_close는 반환값을 따로 가지지 않는다.

- sys\_filesize : 인자로 file이름을 받는다. 인자로 받은 이름을 가지는 file을 찾아 해당 파일의 size를 반환한다.

- sys\_seek : 인자로 file이름과 정수를 입력받는다. 인자로 받은 이름을 가지는 file에서 현재 처리를 하고 있는 위치로 위치를 이동시킨다.

- sys\_tell : 인자로 file이름을 입력받는다. 인자로 받은 이름을 가지는 file에서 현재 처리를 하고 있는 위치를 반환한다.

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

- Lock : 이번 프로젝트에서는 file과 관련된 system\_call을 다룬다. 하나의 file을 수정하거나 읽는 가정에서 동시에 여러 개의 process가 접근한다면 race condition문제가 발생한다. 이를 방지하기 위해서 lock을 사용한다. 구현되어 있는 lock\_acquire과 lock\_release 함수를 이용해서 lock을 구현한다. lock\_acquire함수를 호출하면 현재의 File에 다른 process가 접근하지 못하도록 하고, 이후 다시 lock\_release함수를 호출하면 다시 다른 process가 해당 file에 접근할 수 있도록 해준다.

- Semaphore : Process가 execute또는 start 될 때 semaphore 변수를 사용한다. semaphore를 사용해서 Filesystem에서 Synchronization을 구현한다. thread.h 파일의 struct thread 구조체 내부에서 file\_semaphore등을 선언해 process.c 파일에서 sema\_up과 sema\_down을 이용해 Semaphore를 구현하였다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

11/9 ~ 11/10 : syscall.c 내부에서 system call 구현

11/11 ~ 11/12 : Synchrnonization (lock, semaphore) 구현

11/13 : 오류 수정 및 디버깅

11/14 : 보고서 작성

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

**(1) File Descriptor**

**-** file descriptor를 구현하기 위해서는 thread.c 와 thread.h를 수정해야 한다. 우선 thread.h 내부에 있는 struct thread 구조체 내부에 struct file \*fd[128]을 추가해준다. 이는 struct file 구조체를 자료형으로 하는 1차원 배열이다. 이 크기 128짜리의 일차원 배열이 File Descriptor 역할을 해줄것이다. 이후 thread.c 에서 init\_thread 함수에서는 fd 배열의 모든 값들을 NULL로 바꿔준다.

**(2) System Call**

- syscall.c 내부에서 system call들을 구현해준다. \*(uint32\_t\*)(f->esp)값이 갖는 값에 따라 서 create, remove, open, close, filesize, read, write, seek, tell system call을 구현한다. 자세 한 구현 과정은 아래에서 설명하겠다. 또한 이미 구현되어 있는 read와 write system call 부분에 file에 대한 read 와 write또한 추가해준다.

**(3) Synchronization**

- 우선 exception.c 파일내부에서 유효성검사를 하는 코드를 추가해주었다. 원래 !user 와 is\_kernel\_vaddr(fault\_addr)가 존재했는데 여기에 추가해 만약 not\_present값이 true라면 system\_exit(-1) 으로 system을 exit하는 코드를 추가해주었다.

thread.h에서 synchronization에서 필요한 여러 변수들을 새로 선언해주었다. 우선 lock 을 위해 사용될 struct lock file\_lock을 선언해주고 struct thread 구조체 내부에 비정상적 인 종료를 확인해주는 check변수, file 에 관해서 semaphore를 실행해줄 struct semaphore file\_semaphore, 현재 process의 parent process를 가리지는 struct thread \*parent, 현재 실행중인 process가 다루고 있는 file을 가리키는 struct file\*current\_file을 선언해준다.방금 언급한 모든 변수들은 thread.c 내부에 init\_thread 함수에서 초기화 시 켜준 뒤, 사용해준다.

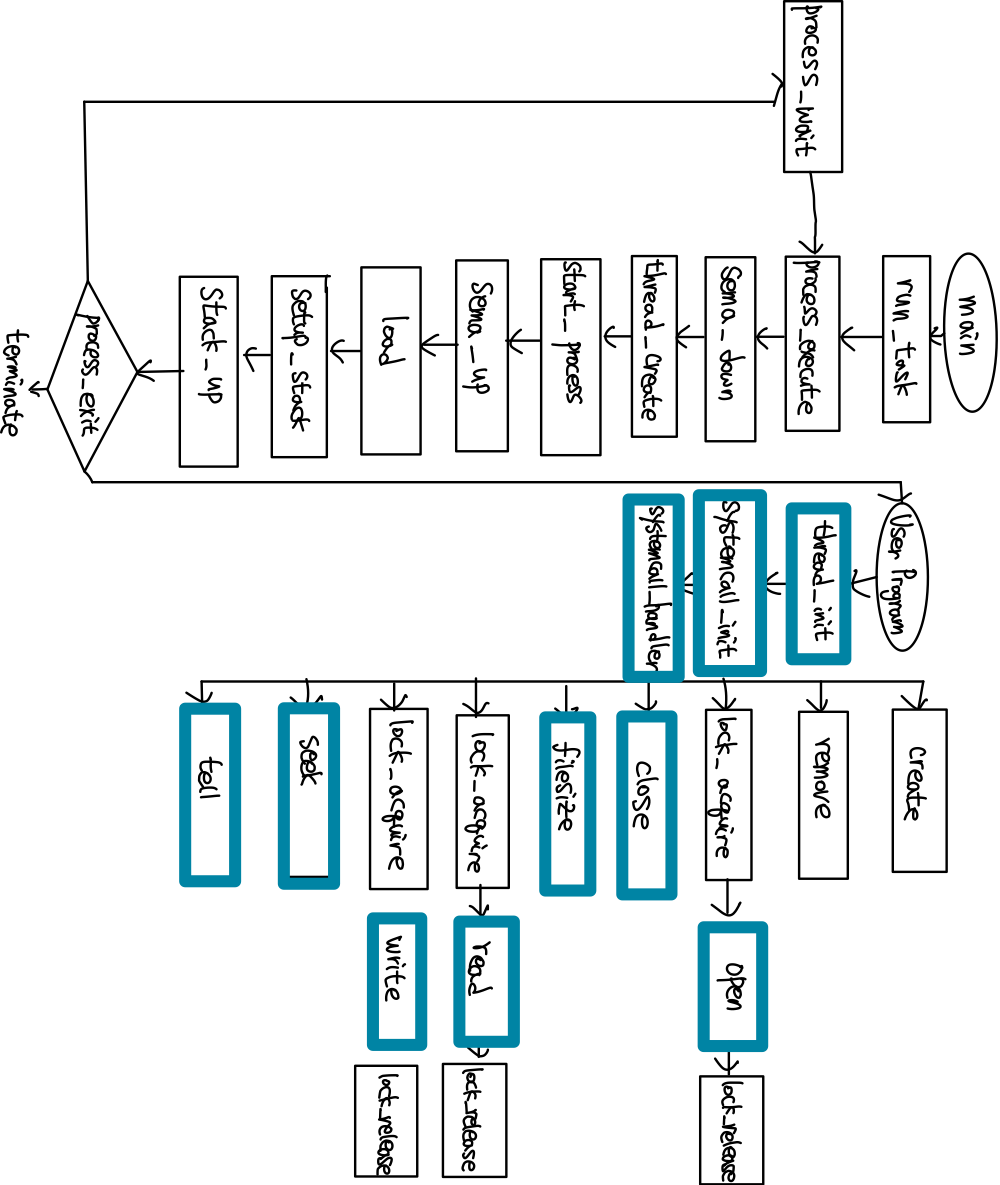
process.c 파일 내부에서 load함수를 수정한다. 성공적으로 load 되었을 경우 success변 수를 만들어 true를 저장한다. 함수 마지막에서 Success변수가 true값을 가질 경우 file에 write하는 것을 deny해준다. process\_execute 함수와 start\_process함수에서 file\_semaphore변수를 사용해서 file을 처리할 때 발생하는 race condition 문제들에 대해 서 synchronization 을 진행해준다. 또한 check 변수를 활용해 비정상적인 종료에 대해서 process\_wait을 시킬지를 결정해준다.

syscall.c 파일에서는 system call을 작성할 때 lock\_acquire와 lock\_release를 사용해서 synchronization 기능을 구현한다. lock\_acquire의 경우 해당 파일을 다른 process들이 건 드리지 못하게 하는 함수이다. lock\_release의 경우 다시 다른 process들이 해당 파일을 처리할 수 있도록 하는 함수이다. 이 두 함수를 이용해 2개 이상의 프로세스가 같은 파 일을 처리하는 것을 방지하는 코드를 작성한다.

1. **연구 결과**
   1. **Flow Chart**

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

1. File Descriptor



2. System Call

텍스트이(가) 표시된 사진

자동 생성된 설명

3. Synchronization

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

* 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.h 파일 내부의 struct thread 내부에 struct file \*fd[128]을 선언해 file descriptor 배열을 만들어주었다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

struct thread 내부에 fd배열 말고 4개의 변수를 추가로 선언했다. 각 변수에 대한 설명은 위 3번에서 했으므로 생략하겠다.

텍스트이(가) 표시된 사진

자동 생성된 설명

thread.c 내부에 init\_thread파일에서 0~127 인덱스이 모든 fd값들을 NULL로 바꾸고, file\_semaphore, parent, current\_file, check 변수를 전부 초기화 해주는 과정을 가진다.

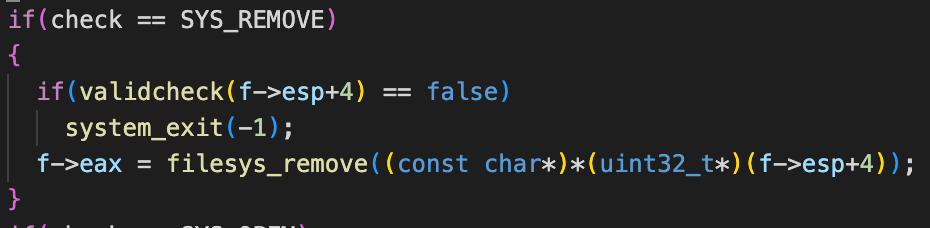
2)System Call

syscall.c 파일 내부에서 여러 system call을 구현하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

f->esp+4, f->esp+8 에 대해서 유효성 검사를 해준 뒤, filesys.c 파일에 구현되어 있는 filesys\_create함수를 실행시킨다.



f->esp+4 에 대해서 유효성 검사를 하고 filesys.c 파일에 구현되어 있는 filesys\_remove 함수를 실행시킨다.

텍스트이(가) 표시된 사진

자동 생성된 설명

f->esp+4 에 대해서 유효성 검사를 한 뒤, filesys.c에서 구현되어 있는 filesys\_open 함수를 실행시켜 open\_file에 open한 file을 저장한다. 이후 fd배열을 3부터 127까지 돌면서 비어 있는 곳에 open\_filed을 넣어준다. 만약 넣는 것에 실패했거나 open\_file이 NULL이라면 -1을 반환한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

f->esp+4에 대해서 유효성 검사를 하고 file.c에 구현되어 있는 file\_close 함수를 실행시킨다. 인자로는 현재 thread가 다루고 있는 file을 넣어준다. 넣어주었다면 해당 file의 fd배열 값을 NULL로 바꿔준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

filesize system call은 f->esp+4 에 대해서 유효성 검사를 하고 file.c 내부에 file\_length함수를 실행시킨다. 인자로는 현재 thread가 다루고 있는 file을 넘겨준다.

seek system call은 f->esp+4, f->esp+8 에 대해서 유효성 검사를 하고 file.c 내부에 file\_seek 함수를 실행시킨다. 인자로는 현재 thread가 다루고 있는 file과 f->esp+8을 넘겨준다.

tell system call은 f->esp+4 에 대해서 유효성 검사를 하고 file.c 내부에 있는 file\_tell 함수를 실행시킨다. 인자로는 현재 thread가 다루고 있는 file을 넘겨준다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

Project1에서 구현했었던 sys\_read 파트를 수정하였다. fd가 3이상인 경우 file을 읽는 것이므로 해당 코드를 추가하였다. file.c 내부에 구현되어있는 file\_read 함수를 실행시켰다. 인자로는 현재 thread가 다루고 있는 file과 f->esp+8, f->esp+12 를 건내준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

write system call도 read와 비슷하다. file\_d값이 2이상인 경우 file을 write하는 것이다. 이를 위해서 file.c 내부에 있는 file\_write 함수를 실행시킨다. 인자로는 현재 thread가 다루고 있는 file, f->esp+8, f->esp+12를 건내준다.

3) Synchronization

if (!user || is\_kernel\_vaddr(fault\_addr)||not\_present) {

exception.c 내부에서 system\_exit을 하는 조건에 not\_present를 추가해준다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

process.c 내부에서 load 함수내부에 success가 true인 경우, 즉 load가 successful하게 이루어진 경우 file\_deny\_write함수를 이용해서 더 이상 file에 적을 수 없게 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

process\_execute 함수에서 file\_semaphore를 sema\_down을 사용하였다. sema\_down을 사용해 parent process가 비정상적으로 끝나는 경우, 현재 process가 처리하고 있는 작업을 비정상적으로 끝마치지 않도록 하게 해준다. 또한 현재 process가 비정상적으로 종료되는 것을 막기 위해서 chk변수를 사용하였다. chk변수가 1인 경우 process\_wait함수를 호출한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위 chk 변수를 start\_process에서 true로 바꿔주고 sema\_down한 것을 start\_process 함수에서 sema\_up 해준다.

위에서 첨부한 syscall.c 함수들을 확인해보면 file에 접근할 때 lock\_acquire한 뒤 file에 접근하는 것을 볼 수 있다. lock\_acquire 를 함으로써 현재 process가 처리하고 있는 file에 다른 process들이 접근하지 못하게 한다. file 처리가 끝난다면 다시 lock\_release해서 다른 process들이 다시 file에 접근하도록 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

system\_exit 에도 코드를 추가해줘야 한다. 프로그램이 돌아가는 동안 설정해줬던 fd배열들을 3~128까지 돌면서 전부 close해주고 배열 값을 NULL로 바꿔준다.

* 1. **시험 및 평가 내용**

텍스트이(가) 표시된 사진

자동 생성된 설명**텍스트이(가) 표시된 사진

자동 생성된 설명**