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

담당 교수/분반: 김영재/1

이름/학번: 백종환/20201587

개발 기간 :24.10.28 ~ 11.04

1. **개발 목표**

* file system과 관련한 system call을 구현한다. File descriptor를 통해 파일 접근을 관리하고 여러 process에서 동일한 파일에 접근할 때 critical section을 보호하기 위해 synchronization을 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
      1. **File descriptor**

* 각 process들이 파일에 접근할 때 file descriptor를 이용하면 간단하게 숫자로 파일에 접근할 수 있고 이를 통해 효율적으로 open, read, write등을 할 수 있다. 또한 각 process마다 file descriptor table을 가지고 있어 각 process들이 같은 파일을 읽기모드, 쓰기 모드 등 동시에 접근이 가능하다.
  + 1. **System call**
* User program이 직접적으로 file system에 access할 수 없기에 이를 위해서 필요한 것으로 이를 통해 kernel mode를 이용하여 file을 열고 닫고 쓰고 읽는 등의 요청을 처리할 수 있다. 이를 통해 user program이 파일 입출력 등의 작업을 할 수 있다. Create, open, close등의 system call에 대한 syscall handler를 구현한다.
  + 1. **Synchronization in Filesystem**
* 만약 여러 process에서 한 파일에 접근할 때 모두 읽기를 수행한다면 데이터가 변하지 않기에 괜찮지만 만약 한 process에서 write를 하는 동안 다른 process에서 read를 한다면 데이터 충돌 등의 문제가 발생할 수 있다.
* Lock, semaphore같은 synchronization mechanism을 이용하면 critical section에 동시에 한 process만 접근 가능하게 하여 데이터 충돌 등의 문제를 해결할 수 있다.
  1. **개발 내용**
     1. **File descriptor**
* 각 process의 fd table을 array로 사용할 것이다. Array의 index의 경우 system call의 argument로 넘겨줄 fd number가 되고 array의 저장된 값은 file 구조체이다. Array를 사용하는 이유는 array의 경우 index를 활용할 경우 그에 해당하는 file을 찾는데 O(1)의 시간이 걸리므로 빠른 조회가 가능하고 구현이 간단하다. 만약 list를 사용한다면 fd번호에 해당하는 file을 조회할 때 O(n)의 시간이 걸리므로 비효율적이다.
  + 1. **System call**
* create: 인자로 받은 file name과 initial\_size로 사이즈에 해당하는 file을 만든다. 성공하면 true 실패하면 false를 반환한다. open은 하지 않는다.
* remove: 인자로 받은 file에 해당하는 file을 삭제한다. 파일이 open되어있든 close되어있든 상관없다
* open: file을 open하여 fd number에 해당하는 것을 return한다. file이 안 열리면 -1을 return
* close: fd에 해당하는 file을 close함 process가 종료될 때 열린 file들을 모두 닫아야함
* filesize: 입력받은 fd로 열린 file의 size를 byte로 return한다.
* read: fd에 해당하는 file로부터 size에 해당하는 byte를 buffer로 읽는다. 실제로 읽은 byte를 반환한다. end of file이면 0, read가 불가능하면 -1을 return한다.
* write: buffer에서 fd에 해당하는 file에 size에 해당하는 byte를 write한다. 실제로 쓴 byte만큼을 return한다. eof에서는 0을 return한다.
* seek: fd에 해당하는 file에 read or write 위치를 position으로 변경함 position은 byte단위임
* tell: fd에 해당하는 file의 read, write pointer가 위치한 position에 해당하는 byte를 return함
  + 1. **Synchronization in Filesystem**
* 만약 한 process가 write를 하는 중에 다른 process가 read나 write를 한다면 데이터가 손상될 수 있으므로 쓰기를 수행할 때는 쓰는 process만 file에 접근이 가능하도록 lock을 사용한다
* parent가 child process를 execute할 때 child process를 시작하기위해 file을 memory에 load해야하는데 이 때 synchronization을 위해 semaphore가 필요하다

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* **개발기간: 24.10.28 ~ 11.04**
* 10.28 ~ 10.31: file descriptor 선언과 synchronization을 고려하지 않은 기본적인 system call 구현
* 11.01 ~ 11.04: Synchronization in Filesystem semaphore, lock을 이용하여 구현함
  1. **개발 방법**
     1. **File descriptor**
* src/thread/thread.h의 thread 구조체
* strcut \*file fd\_table[FD\_MAX]

file 구조체를 담는 array fd\_table을 선언하고 총 128개의 File descriptor가 들어 갈 수 있도록 하였다. #define FD\_MAX 128로 선언하였다. 각 process마다 fd\_table을 가질 수 있도록 하였다.

* Int fd\_num: 현재 process가 현재 사용하는 fd의 개수를 저장하였다.
  + 1. **System call**
* src/userprog/syscall.c에 아직 구현하지 않은 system call을 처리하기 위한 handler에 각각의 syscall을 처리하기 위한 함수를 선언하였다.
* 또한 read, write를 stdin, stdout이 아닌 file에도 적용할 수 있도록 함수를 수정하였다.
  + 1. **Synchronization in Filesystem**
* src/userprog/syscall.c에 struct lock file\_lock을 선언
* open, write, read할 때 한 번에 한 process만 실행할 수 있도록 syscall handler 함수 안에 lock을 사용하였다
* child process를 execute할 때 file이 memory에 load되기전에 parent가 기다려야 하므로 semaphore를 이용함
* thread 구조체에 struct semaphore load\_sema를 추가
* src/userprog/process.c에 있는 process\_execute와 start\_process에서 semaphore를 이용하여 load할 때 오류가 없도록 한다.

1. **연구 결과**
   1. **Flow Chart**
      1. **File descriptor**

**도표, 평면도, 라인, 기술 도면이(가) 표시된 사진

자동 생성된 설명**

* + 1. **System call**

**도표, 라인, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명**

**도표, 라인, 그래프, 평행이(가) 표시된 사진

자동 생성된 설명**

* + 1. **Synchronization in Filesystem**
* **System call**

**도표, 라인, 평면도, 평행이(가) 표시된 사진

자동 생성된 설명**

* **file load**

**도표, 텍스트, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

* 1. **제작 내용**
     1. **File descriptor**
* src/threads/thread.h의 thread 구조체 fd\_table, fd\_num추가

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* 하나의 thread마다 최대 File descriptor의 개수를 128개로 설정하고 array로 fd\_table을 선언하였다.
* 현재 thread에서 open하여 fd를 저장한 개수를 fd\_num에 저장하였다.
* thread.c 의 init\_thread function에서 다음을 추가하였다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

* thread를 만들 때 모든 fd\_table을 NULL값으로 초기화하고 stdout, stdin을 위한 fd는 0, 1로 정해져있으므로 fd\_num값을 2로 초기화해준다.
  + 1. **System call**
* **src/userprog/syscall.h**

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

* pintos manual에 맞게 project2에 구현해야하는 syscall을 handle하기 위한 함수를 선언함
* **src/userprog/syscall.c**
* 파일과 관련한 systemcall은 src/filesys/file.c or filesys.c에 있는 함수를 이용한다.

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

* create, remove의 경우 file의 이름을 받아 file을 만들거나 삭제하는 함수이다 file의 유무에 따라 성공여부과 반환된다. 둘 다 is\_vaild\_user\_pointer에서 null인지 확인한다
* filesize와 seek는 fd number를 입력 받아서 file\_length, file\_seek를 이용하여 file size를 반환받거나 file offset을 변화시킨다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

* tell의 경우는 file의 offset을 반환받는다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

* close의 경우에는 fd를 입력받아 fd에 해당하는 file을 close하고 fd\_table을 null한다.

**텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명**

* open의 경우에는 file 이름을 입력받아 fd\_table에서 비어있으면 fd number를 지정해주는 syscall function이다. fd number를 반환한다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

* read, write 모두 fd와 buffer, size를 입력받는데 read의 경우 buffer에 file의 내용을 size만큼 읽어 저장하고 write의 경우 buffer에 해당하는 내용을 size만큼 file에 write하는 것이다. Read할 때 fd가 stdin에 해당하는 0일 경우에는 input\_getc를 이용하여 buffer에 저장한다. Write의 경우에는 fd가 1일 때는 putbuf을 이용한다.
* 모든 syscall 함수 중 fd를 인자로 받는 함수는 fd의 범위가 유효한지 확인하여야 한다. 즉 fd\_table에 fd가 mapping이 되어있어야한다.
* fd < 2 || fd > FD\_MAX, thread\_current()->fd\_table[fd] != NULL이어야한다.
  + 1. **Synchronization in Filesystem**
* **System call**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* write, read, open 할 때 여러 process가 동시에 접근한다면 데이터 충돌 등의 문제가 생길 수 있으므로 src/threads/synch.h에 선언된 lock을 이용하기 위해 include하였다. Lock구조체를 file\_lock으로 선언하고 syscall\_init에서 lock\_init또한 해주었다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

* open할 때 여러 process가 한 번에 같은 file을 연다면 문제가 생길 수 있으므로 filesys\_open 전에 lock을 걸어 한 process씩 열 수 있도록 하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* write와 read의 경우에도 파일을 쓰거나 읽기전에 lock을 걸어 하나의 한 process만 사용할 수 있도록 하였다.
* **file load**

**텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명**

* thread 구조체에 parent thread가 child thread를 execute하고 child thread가 file을 memory에 load할 때 parent process가 이를 기다리지 않고 수행하면 파일 동기화에 문제가 생길 수 있으므로 child thread가 load할 때에는 semaphore를 이용하여 parent가 wait할 수 있도록 하였다.
* Process.c의 process\_execute()함수이다.
* 텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

  자동 생성된 설명
* parent process가 thread\_create를 통해 child thread를 만들면 이 tid를 입력받아 seam\_down을 선언하여 child thread가 load를 완료할 때 까지 대기한다.
* Process.c의 start\_process function이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* 만들어진 child thread가 수행하는 함수로 load가 완료되면 sema\_up을 통해 parent thread가 대기를 해제하도 록한다.
* **executable file write**

****

* executable file write는 현재 실행하고 있는 파일에 대한 write를 막는 것이다. 이를 위해 처음에는 thread 구조체에 exec\_file만 선언하고 exec하는file을 exec\_file에 저장하고 deny\_write를 하였다. 그리고 나서 process\_exit 할 때 allow\_write를 call하였다. 하지만 이 방식의 문제점은 parent thread가 child thread를 exec하고 child thread가 종료를 해도 parent thread는 이 exec한 file에 대한 쓰기 권한을 회복하지 못 한다는 것이었다.
* 이를 해결하기 위해 thread 구조체에 parent process가 execute하는 file인 parent\_exec\_file, 그리고 현재 thread가 실행하는 파일인 now\_exec\_file을 선언하였다.
* 텍스트, 폰트, 스크린샷이(가) 표시된 사진

  자동 생성된 설명
* Process\_execute할 때 execute하는 파일을 parent thread에 저장하는것이다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* Child thread가 start할 때 parent\_exec\_file에 대한 정보를 now\_exec\_file에 저장한다.

Process\_wait

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

* 자식 process가 종료될 때 wait에서 allow\_write를 해주어 parent thread가 죽을 때가 아닌 자식 thread가 죽을 떄 쓰기 권한을 회복하였다.
  1. **시험 및 평가 내용**

**스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명**

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명