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

담당 교수 / 분반 : 김영재 교수님 / 2

이름 / 학번 : 홍지연 / 20211608

개발 기간 : 2023.10.26 ~ 2023.10.29

1. **개발 목표**

유저모드에서 실행되는 유저 프로그램 중 파일 시스템을 작동하게 한다. 컴퓨터 내 다양한 파일 관련 시스템 콜들이 수행될 수 있도록 먼저 thread에서 사용하는 파일을 인식하고, 관련 유저 프로그램을 수행하고, 실행하면서 발생하는 시스템 콜들을 handler에서 구현한다. 여러 thread에서 동일한 파일을 동시에 접근하여 일관성이 훼손되는 경우를 제어하기 위해 synchronization을 구현한다.

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

핀토스의 각 thread는 보조기억장치에 있는 file에 접근하기 위한 정보를 관리하기 위해 고유한 file descriptor을 가진다. 따라서 threads/thread.h에 thread 구조체에 file descriptor를 구현하고 이를 통해 각종 파일 시스템과 관련된 시스템 콜을 구현한다. 이 과정에서 file 구조체를 여러 c파일에서 사용하기 위해 filesys/file.c에 있던 구조체를 filesys/file.h로 옮겼다.

1. System Calls

userprog/syscall.c에서 open, close, create, remove, filesize, seek, tell를 구현하고, 프로젝트 1에서 구현했던 write, read, exit을 수정하여 업데이트한다. 이 과정에서 filesys/file.c에 구현된 함수를 활용하였다.

3. Synchronization in Filesystem

핀토스는 여러 thread가 돌아가며 프로그램을 수행하는 multi-thread 체제이다. 따라서 여러 thread가 동일한 파일에 접근하여 일관성 문제가 생길 수 있다. 따라서 여러 thread에 의해 write가 진행되거나, write 도중 read를 수행하거나, read 도중 write되지 않도록 방지한다.

* 1. **개발 내용**

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

각 thread마다 고유한 file descriptor를 가진다. 따라서 thread.h의 thread 구조체에 file\* 자료형의 배열로 file descriptor를 선언하였다. struct file\* fp[128]이므로 최대 128개의 파일을 열어볼 수 있는데, 배열이므로 동시에 여러 파일을 열어볼 때 random access가 쉽다.

1. System Calls: 구현할 각 system call에 대해 간략히 서술

int sys\_open(const char\*): file을 열고 0 이상의 정수가 file descriptor로 반환된다. 실패하면 -1이 반환된다.

void sys\_close(int): file\_close로 file descriptor를 통해 열린 파일을 닫는다.

bool sys\_create(const char\*, unsigned): initial\_size의 크기의 file을 생성하는데 성공하면 true, 실패하면 false를 반환한다.

bool sys\_remove(const char\*): file을 삭제하는데 성공하면 true, 실패하면 false를 반환한다.

int sys\_filesize(int): 열린 파일의 크기를 file descriptor를 통해 byte로 반환한다.

void sys\_seek(int, unsigned): file\_seek으로 file descriptor를 통해 열린 파일에서 position에 따라 다음에 읽거나 쓰여진 다음 byte의 위치를 바꾼다.

unsigned sys\_tell(int): file\_tell로 file descriptor를 통해 열린 파일에서 다음에 읽거나 쓸 위치를 정수로 반환한다.

1. Synchronization in Filesystem: Lock, Semaphore를 어떻게 이용할 수 있는지 각각에 대해 설명

두 개 이상의 thread에서 동일한 파일을 쓰거나 읽을 때 synchronization 문제가 발생한다. 따라서 lock과 semaphore를 사용하여 문제를 해결한다. lock은 0 또는 1의 값만 가질 수 있으므로 한 번에 하나 이상의 critical section, 즉 write나 read가 실행되는 동안 다른 thread에서 write나 read가 수행되는 것을 막을 수 있다. 더 나아가 semaphore를 통해 lock을 확장하여, 문제가 발생하지 않고 여러 thread에서 동시에 read를 가능하게 할 수 있다. lock과 마찬가지로 write의 진행을 막는 것 또는 일반 목적의 mutex에서 read의 개수를 세어 처음의 read일 경우 새로운 write를 막고 마지막 read일 경우 새로운 write을 가능케 하는 방식으로 구현할 수 있다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

20231026: 개발에 필요한 학습 진행

20231027~20231028: 목적에 부합하는 개발 진행

20231029: 보고서 작성

* 1. **개발 방법**
* File Descriptor

thread.h에서 thread 구조체에 file descriptor를 선언해주고, thread.c의 init\_thread()에서 해당 배열의 원소들을 모두 NULL로 초기화해주었다.

또한 file.c의 file 구조체를 다른 파일(syscall.c 등)에서 사용할 수 있도록 file.h로 옮겨주었다.

* System Calls

syscall.c에서 system call 함수들을 구현하고, 해당 함수들의 프로토타입을 syscall.h에 작성하였다. syscall\_handler()에서 switch문 안에 이 명령어들을 추가하고 각 명령어에 맞는 함수를 호출하여 system call을 수행하였다. 각 함수에 필요한 인자들은 project1에서 만든 것과 동일하게 esp를 통해 stack의 argument를 넘겨주는 형태이다. 명령어 실행 전, is\_user\_vaddr을 이용한 함수 check\_address()를 통해 유효성 검사를 수행하였다. 또한 함수 내에서 file descriptor의 유효성 검사를 실행하여 만약 없다면 sys\_exit(-1)을 호출하여 종료하였다. 명령어 수행 후 return 값이 생기면 eax에 저장하였다.

exception.c에서 page\_fault() 내에서 not\_present가 0이 아닐 때 sys\_exit(-1)을 호출하여 종료하는 코드를 추가하였다.

* Synchronization in Filesystem

thread.h에서 thread 구조체에 struct semaphore load\_sema, struct thread\* parent, struct file\* cur\_file, int flag를 추가하였다. load\_sema는 child가 실행 중인지 관리하고, parent는 parent를 가리키고, cur\_file은 현재 실행 중인 file을 가리키고, flag는 비정상적으로 종료된 child를 관리한다. 이후 thread.c의 init\_thread()에서 각 함수들을 모두 초기화하였다.

이후 process.c에서 synchroniation을 위한 코드를 작성하였다. 먼저 process\_execute()와 process\_start에 load\_sema를 통해 child가 끝나거나 load되기 전에 parent가 끝나는 경우 메모리를 정리한다. 비정상적으로 종료된 child는 flag를 이용하여 처리한다. 또한 thread가 실행 중일 때 write을 막기 위해 load()에서 cur\_file에 해당 file을 저장하고, file\_deny\_write로 처리한다.

syscall.c에서는 lock 기능을 구현하여 파일에 대한 동기화를 수행하도록 하였는데, sys\_open(), sys\_read(), sys\_write()가 file\_lock을 통해 동일한 파일이 동시 처리되지 않도록 하는 대상이다. 그리고 sys\_exit() 함수에서 thread의 cur\_file을 닫고 종료하는 코드를 추가한다.

1. **연구 결과**
   1. **Flow Chart**

* File Descriptor

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

thread\_init()에서 file descriptor를 초기화하고, system call에서 독립적인 file descriptor를 사용한다.

* System Calls

텍스트, 도표, 평면도, 직사각형이(가) 표시된 사진

자동 생성된 설명

* Synchronization in Filesystem

텍스트, 도표, 평면도, 평행이(가) 표시된 사진

자동 생성된 설명

* 1. **제작 내용**
* File Descriptor

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

userprog/thread.h의 thread 구조체에 file\* 배열의 fd로 file\_descriptor를 선언하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

userprog/thread.c의 init\_thread()에서 file descriptor인 fd에 대해 배열의 인자들을 모두 NULL로 초기화하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

filesys/file.c에 있는 filesys/file.h로 옮겼다. 이를 통해 file.h를 include하면 file 구조체를 모든 파일에서 자유롭게 사용할 수 있도록 한다.

* System Calls

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

userprog/syscall.h에 구현할 함수들의 프로토타입을 선언하였다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

userprog/syscall.c의 syscall\_handler()의 switch문에 구현한 함수들에 대한 case를 추가하였다. 각 case에서 esp를 통해 stack에 접근하는 인자(check\_address를 통해 user memory access를 검사)와 함께 system call을 처리하는 함수를 호출한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

먼저 file의 user memory access를 재차 검사하고 NULL이 아닌지 확인하고 그러하다면 sys\_exit(-1)를 호출한다. filesys/filesys.c에서 제공하는 filesys\_open 함수를 호출하여 처리한다. 이 함수는 주어진 이름의 file을 여는데, 만약 성공한다면 새로운 file을 return하고 아니면 NULL를 return한다. sys\_open은 filesys\_open의 결과가 실패라면 -1을 return하게 한다. 성공이라면 for문을 통해 NULL인 fd를 찾아 file을 저장하고 해당 index 번호를 return한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

먼저 fd가 NULL이 아닌지 확인하고, NULL이라면 sys\_exit(-1)을 호출하여 종료한다. sys\_close는 file\_close 함수를 통해 file을 닫은 뒤 fd를 NULL로 초기화한다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

먼저 file이 NULL이 아닌지 확인한다. 다음으로 주어진 이름의 file을 initial\_size만큼 만드는 함수인 filesys\_create 함수를 통해 처리한다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

먼저 file이 NULL이 아닌지 확인한다. 다음으로 주어진 이름의 file을 제거하는 filesys\_remove 함수를 통해 처리한다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

먼저 fd가 NULL이 아닌지 확인한다. 다음으로 FILE의 byte size를 return하는 file\_length 함수를 통해 처리한다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

먼저 fd가 NULL이 아닌지 확인한다. 다음으로 file의 시작점에서 새로운 위치로 현재 위치를 set해주는 함수 file\_seek를 통해 처리한다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

먼저 fd가 NULL이 아닌지 확인한다. 다음으로 file의 시작 위치에서 현재 위치까지의 byte offset을 return하는 함수인 file\_tell을 호출하여 처리한다.

다음으로 project 1에서 구현했던 함수에 file system을 처리하는 코드를 추가하여 수정하였다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

read 함수의 경우 먼저 is\_user\_vaddr()을 통해 buffer가 valid한 memory인지, buffer가 NULL이 아닌지 확인했다. 다음으로 fd가 3 이상일 때 file system에 대한 처리를 하였다. fd가 NULL이면 먼저 에러 처리를 하고 buffer로 주어진 size만큼 file을 읽는 함수인 file\_read를 호출하여 처리하였다.

텍스트, 스크린샷, 소프트웨어, 운영 체제이(가) 표시된 사진

자동 생성된 설명

write 함수의 경우, fd가 3 이상이면 file system에 대한 처리를 하였다. fd가 NULL이면 먼저 에러 처리를 한 뒤 buffer로 주어진 size만큼 file에 작성하는 file\_write 함수를 호출하였다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

userprog/exception.c의 page\_fault에 if (not\_present) sys\_exit(-1)을 추가하였다. 이는 page가 not\_present일 때 에러 처리를 해주는 코드로, mapping되지 않은 주소 공간에 접근을 하는 테스트 케이스와 관련되어 있다.

* Synchronization in Filesystem

폰트, 스크린샷, 그래픽, 텍스트이(가) 표시된 사진

자동 생성된 설명

userprog/syscall.c에 lock 구조체 file\_lock을 선언해준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

threads/thread.h의 thread 구조체에 struct semaphore load\_sema, struct thread\* parent, struct file\* cur\_file, int flag를 선언한다. load\_sema는 child가 실행 중인지 관리하고, parent는 parent를 가리키고, cur\_file은 현재 실행 중인 file을 가리키고, flag는 비정상적으로 종료된 child를 관리한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이후 thread.c의 init\_thread()에서 각 변수들을 모두 초기화하였다. load\_sema는 sema\_init()을 통해서, parent는 running\_thread()를 통해서, cur\_file은 NULL, flag는 0으로 설정하였다.

다음으로 userprog/syscall.c에서 synchronization을 위한 코드를 추가한다. 이는 다수의 프로세스가 하나의 file에 동시에 접근하는 것을 막는 file을 동기화하는 코드이며, 이를 위해 sys\_read, sys\_write, sys\_open에서 lock을 구현하였다. 여기서 사용하는 함수는 lock\_acquire, lock\_release인데, 전자는 lock의 semaphore를 sema\_down해주고 후자는 sema\_up한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

함수의 시작부에서 file에 대한 접근 전에 lock\_acquire를 호출하여 lock의 semaphore를 sema down한다. 이후 함수가 종료되기 전에 lock\_release를 호출하여 sema up함으로써 명령어 처리를 보호한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

sys\_exit 함수 내에서 parent가 child보다 빨리 종료되지 않도록 while문을 통해 process\_wait을 호출하여 기다리도록 한다. child가 모두 실행 종료되면 while문을 빠져나가고 cur\_file을 file\_close를 통해 닫고 thread\_exit()을 호출한다.

다음으로 userprog/process.c에서 process 동기화를 구현하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

process\_execute에서 thread가 만들어진 후 thread\_current()->load\_sema를 sema\_down하여 parent가 child보다 먼저 죽지 않도록 만든다. 그리고 current\_thread->child에 대해 flag가 1, 즉 child가 비정상적으로 종료된 경우 process\_wait를 호출하며 return한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

start\_process에서는 load 함수가 끝난 시점에서 current\_thread->parent에 대해 load\_sema를 sema\_up해줌으로써 load가 비정상적으로 종료된 경우를 보호한다. load가 비정상적으로 종료된 경우 flag를 1로 만들고 sys\_exit(-1)을 호출하여 종료한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

load 함수에서는 실행 중인 file에 write 실행을 막고자 load 실행이 성공적일 때 cur\_file에 file을 저장하고 file을 file의 deny\_write 값을 true로 바꾸는 file\_deny\_write 함수를 통해 write을 막는다.

* 1. **시험 및 평가 내용**

패턴, 스크린샷, 패브릭, 예술이(가) 표시된 사진

자동 생성된 설명

multi-oom 외 테스트들은 여러 번 수행하면 한 번 이상 성공했다.

multi-oom 테스트의 목적은 pintos 운영체제의 메모리 누수의 존재 여부를 확인하는 것으로, 메모리 누수를 찾아야 테스트를 통과할 수 있다. 그러나 system call의 구현을 재확인하고 process\_exit()에서 thread\_current()->fd의 모든 인자들에 대해 NULL이 아닌 경우 sys\_close를 수행하여도 테스트를 통과하지 못했다.