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

담당 교수 / 분반 : 김영재 / 2반

이름 / 학번 : 윤다영 / 20170212

개발 기간 : 2020.10.31-2020.11.17

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

이번 프로젝트에서는 파일 입출력을 비롯한 파일 시스템을 구현하여, 파일 시스템과 관련된 시스템 콜 create, remove, open, filesize, read, write, seek, tell, close를 구현한다. pintos에서 제공되는 총 76개의 test 중 저번 프로젝트에서 통과하지 못한 55개의 test들을 모두 통과하도록 프로그램을 작성한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

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

1. File Descriptor

다른 여러 운영체제와 같이 pintos에서는 각 프로세스에서 사용되는 파일일들을 File Descriptor Table에 저장하며, 이 File Descriptor Table에 저장된 File Descriptor 엔트리를 통해 파일에 접근한다. File Descriptor Table은 파일 객체 포인터의 배열로, 예를 들어 open 시스템 콜 호출 시, File Descriptor를 반환하며, close 시스템 콜이 호출되면 File Descriptor Table에서 해당 파일에 대한 File Descriptor 엔트리가 삭제된다. 따라서, 각 프로세스는 File Descriptor Table을 통해 자신과 관련된 파일들을 관리하므로 File Descriptor가 이번 프로젝트에서 구현되어야 한다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

이번 프로젝트에서 남은 모든 test들을 통과하기 위해서는 파일 시스템과 관련된 시스템 콜들을 구현해야 한다. 따라서 파일 시스템과 관련된 시스템 콜 create, remove, open, filesize, read, write, seek, tell, close를 구현하여 pintos의 파일 시스템이 제대로 작동하도록 해주어야 한다. 또, 위의 시스템 콜들을 구현할 때, 잘못된 주소에 접근하거나 파일 동기화 문제를 발생시키려고 할 때에도 프로그램이 비정상적으로 종료되지 않도록 구현해주어야 한다.

3. Synchronization in Filesystem

파일 시스템에서 공유되는 데이터들이 접근되는 critical section같은 경우 한 번에 여러 thread에서 read, write 시스템 콜 등을 사용하여 접근하게 될 경우 여러 thread에서 데이터를 조작하여 데이터가 일관성을 잃게 될 수 있다. 따라서 이 때에 동기화를 해주어 데이터의 모순이 생기지 않도록 thread를 관리해주어야 한다. 따라서 이번 프로젝트에서는 파일 시스템에서 critical section에 접근이 발생할 경우 lock, semaphore 등의 synchronization tool들을 사용하여 이와 같은 동기화를 구현해주어야 한다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

File Descriptor를 구현하기 위해, pintos manual을 참고하여 File Descriptor Table을 array를 사용하여 구현하였다. File Descriptor Table는 파일 포인터를 각 인덱스에 담는 array로, File Descriptor Table에서 실제 파일 포인터는 인덱스 3부터 저장되며, 인덱스 0, 1, 2는 NULL 포인터를 가리키며 File Descriptor Table의 인덱스 0은 STDIN을, 인덱스 1은 STDOUT을, 인덱스 2는 STDERR를 의미한다. array를 사용하여 File Descriptor를 구현했을 때, 각 프로세스에 해당하는 파일들을 쉽게 관리할 수 있으며, 표준 입출력도 가능하고 동기화 구현도 용이하므로 array로 구현한 File Descriptor Table을 사용하여 File Descriptor를 관리할 수 있도록 프로그램을 작성했다.

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)

먼저, create는 주어진 initial\_size 만큼의 크기로, 새로운 파일을 생성해주는 system call이다. remove는 지정된 파일을 제거하는 시스템 콜이다. open은 지정된 파일을 여는 시스템 콜이다. 파일을 여는 데에 성공하면 그 파일에 대한 file descriptor를 생성하고, 해당 프로세스의 file descriptor table에 추가해준다. filesize는 인자로 받은 file descriptor에 해당하는 파일을 찾아 그 파일의 크기를 반환하는 시스템 콜이다. read는 주어진 file descriptor가 표준 입력일 경우 키보드에서 주어진 buffer로 내용을 읽어오고, 만약 주어진 file descriptor가 3 이상이면 buffer로 지정된 파일 내용을 읽어오는 시스템 콜이다. write는 주어진 file descriptor가 표준 출력일 경우 모니터에 주어진 buffer의 내용을 출력하고, file descriptor가 3 이상이면 지정된 파일에 buffer의 내용을 쓰는 시스템 콜이다. seek는 지정된 파일의 위치를 이동하는 시스템 콜이다. tell은 지정된 파일의 위치를 반환하는 시스템 콜이다. close는 지정된 파일을 닫는 시스템 콜이다.

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

파일 시스템에서 critical section을 보호하기 위한 synchronization tool로 먼저 lock을 사용할 경우, critical section에 대한 접근이 일어나기 전에 lock을 걸어주어 현재 어떤 thread가 lock을 획득하여 critical section에 접근하지 않는 상태라면 바로 lock을 획득하고, 다른 thread가 이미 lock을 획득하여 가지고 있는 상황, 즉 critical section에 접근한 상황이라면 그 thread가 critical section에서 빠져나올 때까지 기다렸다가 그 thread가 lock을 release한 후에 기다리고 있던 thread가 lock을 획득하도록 하여 critical section에는 항상 하나의 thread가 있음을 보장해준다. 다음으로, synchronization tool로 semaphore를 사용할 경우, binary semaphore를 사용해준다. 즉 semaphore의 value로는 0, 1만 사용하여 critical section에 한 번에 하나의 thread만 접근 가능하도록 해준다. semaphore 사용 시에도 lock과 마찬가지로 critical section 접근 전에 sema down을 해주어 여러 thread가 critical section에 접근하려고 하면 sleep하게 해준다. semaphore에서 기다리고 있던 thread들은 critical section에 있던 thread가 critical section을 빠져나가면 queue에서 하나씩 sema up을 통해 wake되어 하나씩 critical section에 들어가게 된다. 이와 같은 방식으로 semaphore를 사용하여 file synchronization이 가능하다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

|  |  |
| --- | --- |
| 날짜 | 구현 내용 |
| 2020-11-02 | 전체적인 개발 계획 수립 |
| 2020-11-04 | File Descriptor 구현(자료구조 선택, thread 생성, 삭제 시 조치 포함) |
| 2020-11-05 | system call create, remove, open, filesize, seek, tell, close 구현 |
| 2020-11-08 | system call read, write 구현 |
| 2020-11-10 | file system synchronization 구현(lock 사용) |
| 2020-11-12 | test 수행 및 디버깅 |
| 2020-11-13 | 보고서 작성 |
| 2020-11-15 | 보고서 작성 |
| 2020-11-17 | 마지막 점검 및 과제 제출 |

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

먼저, File Descriptor를 구현하기 위해 pintos에서 프로세스(thread)의 context를 의미하는 src/threads/thread.h의 struct thread의 필드로 file descriptor table와 현재 file descriptor table의 다음 인덱스를 가리키는 fd\_next 필드를 추가해주었다. struct thread에 새로운 데이터 필드를 추가해주었으므로 thread 생성 시 file descriptor table을 초기화해주고, thread 종료 시 file descriptor table에 저장된 열린 파일들을 모두 닫아주도록 수정해주었다. 또한 file system에 관련된 시스템 콜들의 구현을 용이하게 하기 위해 src/userprog/process.c에 주어진 파일을 해당 프로세스의 file descriptor table에 저장하는 함수, 주어진 file descriptor에 해당하는 파일을 검색하여 파일 포인터를 반환해주는 함수, 주어진 file descriptor에 해당하는 파일을 닫고 file descriptor table에서 해당 파일의 엔트리를 지워주는 함수를 추가해주었다. 다음으로, 이번 프로젝트에서 완성해야 하는 파일 시스템과 관련된 시스템들을 구현해주기 위해 src/userprog/syscall.c에서 syscall number에 해당하는 switch문을 수정해주고, 위에서 언급한 추가해주어야 하는 시스템 콜들을 함수로 구현해주었다. 마지막으로 file synchronization을 구현해주기 위해 추가한 시스템 콜들 중에서도 read, write에서 lock을 사용해주었다. 또, src/userprog/process.c에서 load 시에 file을 열 때 데이터가 변경되지 않도록 lock을 사용해서 critical section 문제를 예방해주었다.

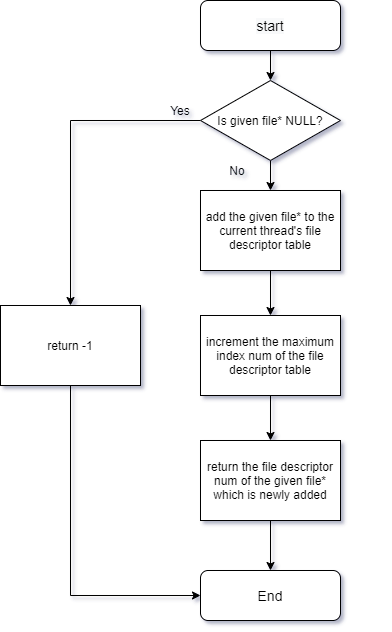
1. **연구 결과**
   1. **Flow Chart**

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

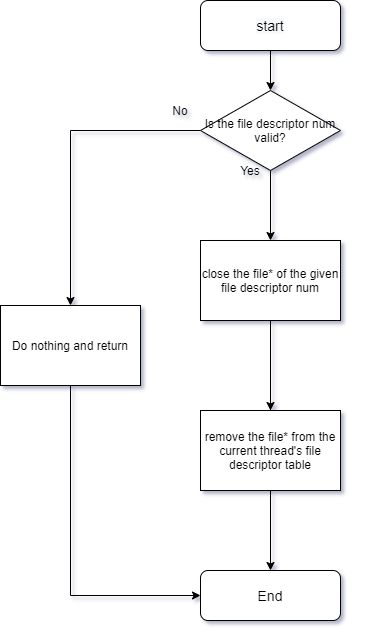
**1. file descriptor 구현**

file descriptor 구현에서 핵심적인 역할을 하는 process\_add\_fd 함수와 process\_file\_close에 대한 flow chart는 다음과 같다.

- process\_add\_fd



- process\_file\_close

****

**2. system call**

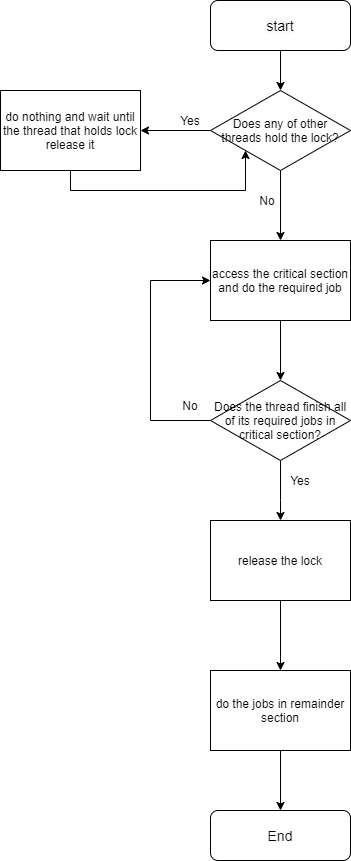
(read, write에서 file synchronization(lock 사용)의 내용은 2.에서 생략 후 3.에서 표현)

컴퓨터, 키보드이(가) 표시된 사진

자동 생성된 설명

**3. Synchronization in Filesystem**

(critical section에 접근하는 모든 함수의 경우의 공통적인 synchronization 알고리즘)



* 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

가장 먼저 File Descriptor를 구현하기 위해 struct thread에 file descriptor table 필드를 새로 추가하였으므로 src/threads/thread.c의 thread\_create 함수에서 새로 thread를 생성할 때에 file descriptor table의 모든 인덱스를 초기화해주고, fd\_next를 3으로 설정해준다. 또, file descriptor와 file system에 대한 시스템 콜들과의 호환이 더 쉽도록 src/userprog/process.c에서 새로운 함수 process\_add\_fd, process\_search\_file, process\_file\_close를 정의하여 사용하였다. process\_add\_fd 함수는 파일 포인터를 인자로 받아 현재 thread의 file descriptor table에 추가해주는 함수로, 추가한 후에는 현재 thread의 next\_fd를 1 증가시켜주고 주어진 파일 포인터의 file descriptor 번호를 반환하도록 하였다. process\_search\_file 함수는 인자로 주어진 file descriptor 번호가 유효하다면, 해당 파일을 찾아서 그 파일의 포인터를 반환해주는 함수이다. process\_file\_close는 주어진 file descriptor 번호가 유효하다면 해당 파일을 닫은 후, 파일 포인터를 file descriptor table에서 지우고, 유효하지 않다면 바로 리턴하는 함수이다. 다음으로 구현한 file descriptor에 대한 내용을 바탕으로 file system에 관련된 시스템 콜들을 구현하였다. 먼저 pintos/src/userprog/syscall.c에 새로운 시스템 콜에 사용될 함수들을 정의해주었다. 구현에는 src/filesys/filesys.h와 file.h의 함수들을 사용하였다. create 함수에서는 주어진 파일 포인터가 유효할 경우 filesys\_create 함수를 사용하여 새로운 파일을 생성하도록 하였다. remove 함수에서는 주어진 파일 포인터가 유효할 경우 filesys\_remove 함수를 사용하여 파일을 삭제하였다. open 함수에서는 주어진 파일 포인터가 유효할 경우 filesys\_open 함수를 사용하여 파일을 열고 process.c의 process\_add\_fd를 사용하여 파일을 file descriptor table에 넣어주고, 할당 받은 파일의 file descriptor 번호를 반환한다. filesize 함수는 주어진 file descriptor 번호가 유효할 경우에 해당 file descriptor 번호의 파일의 길이를 file\_length 함수를 사용하여 구한 후 반환한다. seek 함수는 주어진 file descriptor 번호가 유효할 경우 file\_seek 함수를 사용하여 주어진 position으로 파일의 위치를 이동시켜준다. tell 함수는 주어진 file descriptor 번호가 유효할 경우 file\_tell 함수를 통해 해당 파일의 위치를 알려준다. close 함수는 process.c에서 정의한 process\_file\_close 함수를 호출하여 주어진 file descriptor 번호에 해당하는 파일을 닫고 file descriptor table에서 지운다. 마지막으로 file system에서의 synchronization을 구현하기 위해, read, write 시스템 콜 함수, process.c의 load 함수에서 lock을 사용해주었다. read 함수는 주어진 file descriptor 번호가 표준 입력일 경우 키보드에서 입력을 받아 buffer에 저장한다. 하지만 실제 파일을 read할 때는 file\_read 함수 실행 전에 lock\_acquire(&lock)의 형태로 다른 thread가 lock을 가지고 있지 않은 경우에만 file\_read가 수행되도록 해준다. 또, file\_read 수행이 완료된 후에는 다른 thread에서 다시 lock을 잡을 수 있도록 lock\_release(&lock)을 호출해 lock을 놓아준다. write 함수의 경우에도 file descriptor 번호를 확인한 후 표준 출력이라면 모니터에 buffer의 내용을 출력하고, 실제 파일을 write해야 하는 경우에는 file\_write가 수행되기 전에 lock\_acquire(&lock)을 호출하고, file\_write 수행이 완료된 후에는 lock\_realease(&lock)을 호출한다. 또한, synchronization을 위해서는 현재 실행 중인 파일의 경우 데이터의 변경이 이루어지지 않아야 한다. 따라서 src/userprog/process.c에 정의된 load 함수에서 실행할 파일을 filesys\_open 함수를 통해 열기 전에 lock\_acquire(&lock)을 호출하여 lock을 걸어준 후, 파일을 연 후에는 file\_deny\_write를 호출하여 이 파일에 대해 데이터가 변경되지 않도록 막아준다. file\_deny\_write 함수 호출 후에는 lock\_realease(&lock)을 호출하여 다시 lock을 놓아준다. 이와 같이 구현한 후 test를 수행했을 때 ‘read-bad-ptr’ test를 통과하지 못했다. 따라서, read, write 시스템 콜 함수에서 가장 처음에 인자로 받은 buffer의 주소가 유효한지 is\_valid\_user\_vaddr 함수를 호출하여 확인한 후 유효한 주소에 대해서만 read, write가 수행되도록 수정해주었다.

* 1. **시험 및 평가 내용**
* make check 수행 결과를 캡처하여 첨부

텍스트이(가) 표시된 사진

자동 생성된 설명