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

담당 교수 / 분반 : 김영재 교수님 / 1분반

이름 / 학번 : 조현지 / 120270273

개발 기간 : 14일

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

**파일 시스템을 구현하기 위한 시스템 콜 함수를 구현한다. 구현한 시스템 콜을 바탕으로 파일 동기화 기법이 적용된 파일 시스템을 구현한다.**

1. **개발 범위 및 내용**
   1. **개발 범위**

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

1. File Descriptor

프로세스가 사용하려는 파일에 대한 읽고, 쓰기를 효율적으로 제어하기 위한 자료 구조. 파일 디스크립터를 순회하면 프로세스가 작업하려는 파일이 무엇인지 파악 가능. 프로세스가 개별 파일의 상태 (open(), close(), read(), write())를 효율적으로 관리할 수 있다.

2. (이번 프로젝트에서 구현해야 하는) System Calls

create(), remove(), filesize(), open(), read(), write(), tell(), seek(), close()

파일 생성, 삭제, 읽고, 쓰기에 관련된 시스템 콜 함수이기 때문에, 파일 시스템을 동작하기 위해 기본적으로 정의해야 한다.

3. Synchronization in Filesystem

동일한 파일에 대해 동시에 write를 하거나, 한 쪽은 read를 하고 다른 한 쪽은 write를 하려는 동작을 막기 위해 동기화가 필요하다.

* 1. **개발 내용**
* 아래 항목의 내용만 서술

1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술

자료 구조 : 리스트

선택 이유 : for문을 통해 리스트의 처음부터 끝까지 확인해야 하는 단점이 있지만, 프로세스의 동작에 따라 동적으로 요소를 삽입하거나 삭제할 때 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)

create() : 매개변수로 넘겨받은 파일 이름을 바탕으로, 디렉토리 내에 존재하는 파일을 찾고 이를 파일 객체로 만든다.

remove() : 파일 디스크립터에서 해당 객체를 삭제한 뒤, 삭제가 정상적으로 완료된 경우 true를 반환한다.

filesize() : 현재 스레드의 파일 디스크립터 리스트에서 매개변수로 념겨받은 fd와 동일한 값을 가지는 파일 디스크립터 구조체를 찾는다. 파일 디스크립터 구조체가 가리키는 파일의 크기를 반환한다.

open() : 매개변수로 넘겨받은 파일이 존재하는지 확인한 뒤, 파일 디스크립터의 빈 공간에 push한다.

read() : fd가 0이면 표준 읽기로 동작한다. 아니라면, 파일 디스크립터에서 매개변수로 넘겨받은 fd를 가지는 파일 디스크립터 구조체를 찾는다. 찾은 파일 디스크립터가 가리키는 파일을 가져온 뒤, 읽어 들인 요소의 길이만큼에 대해 반환한다..

write() : fd가 1이면 표준 쓰기로 동작한다. 아니라면, 파일 디스크립터에서 매개변수로 넘겨받은 fd를 가지는 파일 디스크립터 구조체를 찾는다. 찾은 파일 디스크립터가 가리키는 파일을 가져온 뒤, 입력 받은 요소의 길이만큼에 대해 반환한다.

tell() : 전달받은 번호에 해당하는 파일 디스크립터 객체를 통해 파일 객체를 찾는다. 이후, 해당 파일의 position을 확인한다.

seek() : fd를 번호로 가지는 파일 디스크립터를 찾고, 해당 객체가 가리키는 파일을 찾는다. 파일 객체 내에 저장된 위치를 반환한다.

close() : 전달받은 fd를 번호로 가지는 파일 디스크립터 객체를 찾고, 파일 디스크립터 리스트에서 파일 디스크립터를 제거한다.

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

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

|  |  |
| --- | --- |
| 2024.10.20 ~ 2024.10.24 | File descriptor |
| 2024.10.20 ~ 2024.10.24 | System calls : create(), open(), read(), write(), filesize(), remove() |
| 2024.10.24 ~ 2024.10.28 | System calls : tell(), seek(), close() |
| 2024.10.28 ~ 2024.11.05 | Synchronization in Filesystem |

* 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

|  |  |
| --- | --- |
| File Descriptor | <thread.h, thread.c>  - 자료 구조 : struct list file\_descriptor (파일 디스크립터 구조체를 담아두기 위한 리스트 변수), struct list\_elem elem (리스트 요소 변수) 추가하기. Struct file \*execution\_file (현재 실행중인 파일을 가리키는 변수) 추가하기.  - 소스 코드 : thread의 init\_thread()와 thread\_init()에 파일 디스크립터 리스트를 초기화하는 함수 추가하기  <file\_descriptor>  - 자료 구조 : 파일 디스크립터 번호, 파일 정보를 저장해둘 파일 디스크립터 구조체 정의하기  - 함수 : 빈 파일 디스크립터 번호 반환, 표준 입출력과 에러에대한 파일 디스크립터 객체 생성하고 이를 리스트에 넣기, 파일 디스크립터를 리스트에 추가, 파일 디스크립터 번호로 파일 디스크립터가 가리키는 파일 반환하기 |
| System calls | <syscall.h, syscall.c>  -소스코드 :  1. 기존의 read() -> 변경된 read() : fd = 0일 때, 표준 입력인지 특정 파일을 가리키고 있는지 확인하는 로직이 필요하다.  2. 기존의 write() -> 변경된 write() : fd = 1일 때, 표준 출력인지 특정 파일을 가리키고 있는지 확인하는 로직이 필요하다.  3. 그 외의 create, remove, open, filesize, seek, tell, close 시스템 콜 함수를 filesys/filesys.c 파일과 filesys/file.c 파일을 이용해서 구현한다. |
| Synchronization in Filesystem | <thread.h>  -소스 코드: thread 구조체에 struct file \*execution\_file 변수를 추가해, load한 execution file을 가리키게 한다.  <process.c>  -소스 코드 : start\_process에서 실행되는 load 함수가 executable file을 open할 때, thread 구조체의 file 포인터가 해당 파일을 가리키게 한다. Start\_process에 load가 success를 반환하고 나서 실행되는 init\_fd\_list 함수 뒤에 해당 파일을 가리키는 파일 디스크립터를 생성한다. 이후 해당 파일에 대해 file\_deny\_write 함수를 실행한다.  <syscall.c>  -소스 코드 : read나 write 함수에서 파일을 읽으려고 할 때, sema\_up을 해준다. 해당 작업이 끝나고나면 sema\_down을 해준다. |

1. **연구 결과**
   1. **Flow Chart**

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

|  |  |
| --- | --- |
| 1 | 새롭게 생성된 스레드의 파일 디스크립터 리스트를 초기화 |
| 2 | 현재 비어있는 파일 디스크립터 번호 찾기 |
| 3 | 표준 입력, 출력, 에러에 대한 파일 디스크립터가 리스트에 있는지 확인하기 |
| 4 | 파일 디스크립터 번호 0번부터 2번이 존재한다면, 파일 디스크립터 리스트의 비어있는 파일 디스크립터 번호 반환 |
| 5 | 발견한 파일 디스크립터를 바탕으로 새로운 파일 디스크립터 객체를 생성 |
| 6 | 생성된 파일 디스크립터 객체를 스레드의 파일 디스크립터 리스트에 추가 |
| 7 | 이후, 파일 디스크립터 번호를 바탕으로, 파일 디스크립터가 가리키는 파일 반환하기 |

* System Calls

|  |  |
| --- | --- |
| 1 | 작성해둔 파일 디스크립터를 바탕으로 seek(), tell(), filesize() 구현하기 |
| 2 | Filesys API를 이용해서, remove(), create() 함수 구현하기 |
| 3 | Open() 함수는 파일 디스크립터와 filesys\_open() API를 이용해서 파일 디스크립터를 새롭게 생성한 뒤 이에 해당되는 번호를 반환한다. |
| 4 | Read() 함수는 버퍼의 영역을 확인한 뒤 작업을 수행한다. 표준 입력을 먼저 처리한 뒤 파일 읽기를 수행한다. |
| 5 | Write() 함수는 표준 출력인지 먼저 확인하고 나서 파일 출력을 수행한다. |
| 6 | Close() 함수는 파일 디스크립터를 성공적으로 삭제했는지를 먼저 확인한다. 이후에 0, 1, 2번 파일 디스크립터가 불렸을 경우에는 표준 입출력이 아니도록 is\_standard를 false로 만든다. |

* Synchronization in Filesystem

|  |  |
| --- | --- |
| 1 | 파일에 대해 read나 write 함수가 불려졌을 경우 sema\_up을 한다. |
| 2 | 다른 스레드에서 불려진 read/write 함수는 1)에서 실행한 함수가 끝날 때까지 대기한다. |
| 3 | 1)에서의 작업이 끝나면, sema\_down을 한다. |
| 4 | 2)에서 대기중이던 작업이 시작된다. |
| 5 | 현재 실행중인 프로그램의 파일 디스크립터 리스트에 있는 파일에 대해 file\_deny\_write을 해준다. |
| 6 | Read와 write 시스템콜이 진행중일 때 (표준 입력이거나 출력일 때는 관계 없음), 파일에 접근하기 전에 sema\_up을 해준다. 이후, 작업이 끝나고나면 sema\_down을 해줘 다른 스레드가 접근할 수 있게 해준다. |

* 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
* File Descriptor

|  |  |
| --- | --- |
| File descriptor 구현  1. file\_descriptor.h  2. file\_descriptor.c | - file descriptor 구조체를 새롭게 정의: 파일 디스크립터 번호, 파일 디스크립터가 가리키는 파일로 구성된 구조체  - thread의 struct list file\_descriptor : Pintos에 내장된 list 라이브러리를 사용하여 리스트 요소의 추가와 삭제를 용이하게 함.  - assign\_fd : 현재 스레드의 파일 디스크립터 리스트를 3번부터 순환하면서 비어있는 파일 디스크립터 번호 반환를 반환한다. fd = 3에서 시작하며, 가져온 파일 디스크립터 객체의 fd가 예상한 것과 같으면 fd++을 해준 뒤, 다음 파일 디스크립터 객체로 넘어간다. 만약, 두 객체의 fd가 일치하지 않는다면 해당 번호를 가진 파일 디스크립터가 존재하지 않는다는 말이 된다. 따라서 해당 fd를 바로 반환한다.  -init\_fd\_list : stdin, stdout, stderr을 위한 파일 디스크립터 객체를 생성하기 위한 함수이다. 사용자 프로그램이 시작되기 전에 현재 스레드의 파일 디스크립터 리스트 속에 0번, 1번, 2번에 해당하는 파일 디스크립터 구조체를 추가한다 또한, .assign\_fd를 진행하기 전에, 파일 디스크립터 리스트가 비어있을 경우에도 실행한다.  -add\_fd : 현재 스레드의 파일 디스크립터 리스트에 새롭게 생성한 파일 디스크립터 객체를 추가한다.  -create\_fd : 특정 파일을 가리키고 있는 파일 디스크립터를 생성한다.  -find\_pointing\_file : 현재 스레드와 파일 디스크립터 구조체를 매개변수로 넘겨주면, 해당 파일 디스크립터가 가리키는 파일을 반환한다. 만약 파일 디스크립터가 가리키는 파일이 없거나, 파일 디스크립터 리스트에 해당 파일 디스크립터가 존재하지 않는 경우에는 NULL을 반환한다.  -find\_pointing\_fd\_element : 현재 스레드와 파일 디스크립터 번호를 매개변수로 넘겨준다. 파일 디스크립터 리스트를 0번부터 순회하면서, 매개변수로 넘겨준 번호와 일치하는 파일 디스크립터 구조체를 찾고 이를 반환한다. 만약 해당하는 파일 디스크립터 구조체를 리스트 내에서 발견하지 못할 경우 NULL을 반환한다.  -delete\_fd : 표준 입출력과 에러가 있는 0, 1, 2번 파일 디스크립터 구조체의 경우, 단순히 파일 디스크립터가 가리키는 file을 NULL로 바꾼 후 true를 반환한다. 3번 이상의 파일 디스크립터 구조체는 파일 디스크립터 리스트를 0번부터 순차적으로 순회한다. 찾고자하는 파일 디스크립터와 번호가 같은 구조체를 찾았을 경우에만 파일 디스크립터 리스트에서 해당 구조체를 list\_remove 함수를 통해 제거한 뒤 true를 반환한다. 구조체를 찾지 못했을 경우에는 false를 반환한다. |

|  |  |
| --- | --- |
| 이슈1 | init\_fd\_list 함수가 thread의 init\_thread와 thread\_init 함수 내에서 실행되게 하려고 했지만, thread\_current()가 run 상태가 아니라는 출력과 함께 kernel panic이 발생했다. |
| 해결책 | 스레드가 생성 단계에서 스레드 내의 변수에 대해 동적 할당을 시행하려다 발생한 문제라 생각했다. 이를 해결하기 위해서는 동적 할당을 하고자 하는 스레드가 thread\_current()일 때, init\_fd\_list 함수가 실행되어야 한다고 생각했다.  -> 사용자 프로그램이 시작되기 직전인 start\_process() 함수에서 init\_fd\_list 함수를 수행하도록 하였다. Kernel panic 없이 정상적으로 작동되었다.. |

* System Calls

|  |  |
| --- | --- |
| 시스템 콜 구현 | create() : 파일 이름이 비어있거나 공백인지 확인한 뒤, filesys\_create() 함수를 호출한다.  remove() : 파일 이름을 filesys\_remove 함수로 넘긴다. 만약 파일이 삭제된다면, true를 반환한다.  filesize() : 현재 스레드의 파일 디스크립터 리스트에서 매개변수로 념겨받은 fd와 동일한 값을 가지는 파일 디스크립터 구조체를 찾는다. 파일 디스크립터 구조체가 가리키는 파일의 크기를 반환한다.  open() : 파일 이름이 “”이거나 NULL일 경우 -1을 반환한다. 현재 스레드를 thread\_current()를 찾은 뒤, assign\_fd 함수를 통해 현재 비어있는 파일 디스크립트 번호 중 가장 작은 것을 찾는다. 이후, filesys\_open 함수를 통해 파일 이름에 해당하는 file을 찾고 NULL인지 확인한다. 만약 NULL이라면 -1을 반환한다. 이후, 파일, 파일 디스크립트 번호를 create\_fd에 넘겨주면, create\_fd가 새로운 파일 디스크립트를 생성한 뒤 이를 반환해준다. 반한된 파일 디스크립트가 존재하지 않는다면 file\_close 함수를 실행한 뒤 -1을 반환한다. 존재한다면 현재 스레드의 파일 디스크립터 리스트에 해당 파일 디스크립터를 추가한다. 이때, add\_fd에 의해 파일 디스크립터 리스트는 오름차순으로 정렬된다. 이후 파일 디스크립터 번호를 반환한다.  read() : 파일 디스크립터 0번이 is\_standard == true이면, 표준 읽기로 동작한다. 아니라면, 다른 3 이상의 번호를 가지는 파일 디스크립터처럼 파일 디스크립터가 가리키는 파일을 읽어들인다. 만약 표준 출력과 에러를 담당해야하는 1번이나 2번이 매개변수로 들어왔을 경우, exit(-1)을 한다. find\_pointing\_file()이라는 파일 디스크립터 구조체의 함수를 이용해 fd번 파일 디스크립터 구조체를 찾는다. 찾은 파일 디스크립터가 가리키는 파일을 가져온 뒤, 읽어 들인 요소의 길이만큼을 filesys\_read() 함수에 넘겨준다. 해당 함수가 반환하는, 읽어들인 문자의 수를 반환하면 시스템 콜이 종료된다.  write() : fd가 1이고 fd\_element -> is\_standard == true이면 표준 쓰기로 동작한다. 아니라면, 1번 파일 디스크립터가 가리키는 파일을 find\_pointing\_file 함수를 통해 찾는다. 이후 file\_write 함수에 파일, 버퍼, 길이를 매개변수로 넘겨주면 반환되는 입력 길이를 그대로 반환해준다.  tell() : 현재 실행중인 스레드를 반환하는 thread\_ current() 함수와 fd번 파일 디스크립트가 가리키는 파일을 반환하는 find\_pointing\_file() 함수를 실행한다. 이후, file\_tell 함수에 파일 객체를 넘겨 파일의 offset을 반환한다.  seek() : 현재 실행중인 스레드를 반환하는 thread\_ current() 함수와 fd번 파일 디스크립트가 가리키는 파일을 반환하는 find\_pointing\_file() 함수를 실행한다. 이후, 매개변수로 전달받은 파일 위치 position과 파일 객체를 file\_seek 함수에 전달한다.  close() : fd번 파일 디스크립터 객체를 find\_pointing\_ fd\_element 함수를 통해 찾는다. 만일 파일 디스크립터가 없다면 exit(-1)을 한다. 발견되었을 경우, bool을 반환하는 delete\_fd 함수를 실행한다. 만약 delete\_fd가 false라면, 매개변수로 전달받은 fd와 동일한 값을 가지는 fd번 파일 디스크립터 존재하지 않았다는 뜻이기에 exit(-1)을 한다.만약 true를 반환했다면, 먼저 file 객체가 NULL이 아닌지 확인한다. 만약 file 객체가 존재한다면 file\_close() 함수를 통해 파일을 닫는다.  0, 1, 2번 파일 디스크립터는 다른 파일 디스크립터와 동작이 다르기 때문에 각각 if (fd < 3), if (fd > 3)으로 구분하였다.  -0, 1, 2번 파일 디스크립터 : 표준인지 아닌지를 확인하기 위해 파일 디스크립터 구조체의 is\_standard 변수를 확인한다. 만약 is\_standard == true이면, is\_standard 변수를 false로 바꿔준다. False일 경우에는, is\_standard 변수를 앞에서와 같이 true로 바꿔준다. 또한, 표준 파일 디스크립터 구조체를 어떠한 file도 가리키지 않기 때문에 fd\_element -> file = NULL로 바꿔준다.  - 3번 이상의 파일 디스크립터 : 파일 디스크립터에게 할당했던 공간을 해제하기 위해 free(fd\_element)를 실행한다. |

|  |  |
| --- | --- |
| 이슈 | read() 함수가 표준 입출력과 에러를 제대로 처리하지 못해 비정상 종료되었다. |
| 해결책 | 기존의 read()를 변경하면서, if(fd != 0)와 같이 fd에 대한 조건문을 제대로 설정하지 발생한 문제였다.  -> fd에 해당하는 파일 디스크립터 객체가 있는지 확인한다. 아니라면, exit(-1)을 한다.  -> if(fd == 0 && fd\_element -> is\_standard)이면, input\_getc 함수를 통해 length만큼 버퍼의 내용을 읽어들인다.  -> if(fd == 1 || fd == 2)이면 잘못된 fd가 read 함수로 들어온 경우이기에, exit(-1)을 반환한다.  -> 파일 디스크립터 객체가 가리키는 파일이 NULL인지 확인한다. NULL이면 exit(-1)을 한다.  -> 파일 디스크립터가 가리키는 파일에 대해, filesys\_read() 함수를 통해 읽어들인 길이를 반환한다. |

* Synchronization in Filesystem

|  |  |
| --- | --- |
| Deny writes to executable files | start\_process에서 실행되는 load 함수가 executable file을 open할 때, thread의 struct file \*execution\_file 변수가 해당 파일을 가리키게 한다. Start\_process에 load가 success를 반환하고 나서 실행되는 init\_fd\_list(파일 디스크립터 리스트에 0,1,2번 파일 디스크립터를 생성해주는 함수) 뒤에 해당 파일을 가리키는 파일 디스크립터를 생성한다. 이후 해당 파일에 대해 file\_deny\_write 함수를 실행한다.  -> 프로세스가 종료될 때 process\_exit 함수에서 파일 디스크립터 리스트에 있는 모든 파일 디스크립터 객체를 확인한다. 만약 파일 디스크립터 객체가 가리키는 파일이 존재한다면 file\_allow\_write() 함수를 실행시킨다. |

|  |  |
| --- | --- |
| 문제점 | 기존에는 start\_process에서 가리킨 executable file 하나만을 process\_exit 함수에서 file\_allow\_write()를 해줬다.  -> 파일 디스크립터 리스트 내의 모든 파일에 대해 file\_allow\_write 함수가 적용되지 않는 문제가 발생하였다. |
| 해결책 | Process\_exit() 함수에서 파일 디스크립터 리스트 내에 존재하는 모든 파일 디스크립터 객체가 가리키는 파일이 존재한다면 file\_allow\_write을 해준뒤, 닫히지 않은 파일에 대해 file\_close(f)를 해줬다. |

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