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

**(설계 프로젝트 수행 결과)**

과목 명 : [CSE4070] 운영체제

담당 교수 : 교수님

조 / 조원 : 01조, 2016\*\*\*\*, 2013\*\*\*\*

개발 기간 : 2019/11/5 – 2019/11/17

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

**제출일 : 2019년 11월 17일**

**참여 조원 :**

1. **개발 목표**

**지난 프로젝트에서 핀토스 유저 프로그램 구동에 필요한 기본적인 메모리 스텍 구현, 커널 영역과 유저 프로그램 영역의 침범 방지, system call을 통한 표준 출력 및 exit, wait, exec, halt 등 기본적인 기능을 구현하였다. 이번 프로젝트에서는 지난 프로젝트에 이어 유저 프로그램 중 파일을 다루는 open, close, write, read, filesize 등 추가적인 시스템 콜 기능을 구현한다. 그리고 파일 작업 도중 발생할 수 있는 파일 동시 편집과 같은 싱크 이슈를 해결할 것이다.**

1. **개발 범위 및 내용**
   1. **개발 범위**

* **파일관리: 각 스레드가 현재 접근중인 파일 목록을 관리하기 위해 threads/thread.h의 thread 구조체에 struct file\* 형태의 배열을 files라는 이름으로 131개 할당한다.**
* **파일 쓰기금지: 같은 파일을 한 개 이상의 스레드가 읽는 동시에 다른 스레드에서 해당 파일에 내용을 쓸 수 없도록 한다.**
* **Execution & load 싱크 문제: 자식 프로세스가 생성될 때 자식 스레드의 파일 로드가 끝나기 전에 부모 스레드가 wait 상태로 넘어가는 것을 방지한다. 부모 스레드는 자식 스레드의 파일 로드 여부를 확인한 후 wait 상태로 넘어간다.**
  1. **개발 내용**
* **파일 쓰기 금지 및 싱크 문제를 해결하기 위해 두 개의 값만 같는 바이너리 semaphore를 사용하여 spin lock 기법을 이용한다. Binary semaphore는 임의의 sema 값에 대하여 현재 진행중인 프로그램이 특정 신호에 의해 무한루프에 걸려 특정 조건이 만족되기 전까지 프로그램 일시정지하는 기법을 말한다.**

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

|  |  |
| --- | --- |
| **기간** | **구현 목표** |
| **2019/11/4 – 2019/11/10** | **프로젝트 목표 수립 및 문제 정의** |
| **2019/11/11 – 2019/11/14** | **파일처리 관련 시스템콜 기능 구현** |
| **2019/11/15 – 2019/11/16** | **파일 쓰기금지 및 스레드 생성 싱크 조정** |
| **2019/11/17** | **보고서 작성** |

* 1. **개발 방법**

1. **파일관리 구현**

**각 스레드가 어떤 파일에 엑세스하고 있는지를 관리하기 위해 threads/thread.h의 struct thread 에 struct file\* files[131]을 추가한다. 핀토스 매뉴얼에 의하면 각 스레드는 최대 128개의 파일을 열 수 있다. 하지만 일반적인 운영체제에서 파일 구분자인 fd가 0일 때에는 표준 입력, 1일 때 표준 출력, 2일 때 에러를 나타내므로 3번부터 실제 사용자 파일에 접근한다고 가정했다. 따라서 3번 인덱스부터 128개의 파일을 열 수 있도록 131개의 크기로 배열을 할당하였다.**

1. **파일관리 관련 system call 구현**

**파일 시스템에 관련된 open, close, read, write, filesize 등의 시스템콜은 files/filesys.c와 filesys/filesys.c의 함수를 참고하여 구현했다. 다음은 각 기능을 구현하기 위해 사용한 함수이다.**

1. **Open : files/filesys.c filesys\_open()**
2. **Close : filesys/file.c file\_close()**
3. **Create: 새로운 파일을 생성한다. Filesys/filesys.c filesys\_create()**
4. **Remove: 해당 파일을 삭제한다. Filesys/filesys.c filesys\_remove()**
5. **Seek : filesys/file.c file\_seek()**
6. **Tell: filesys/file.c file\_tell()**
7. **Filesize : filesys/file.c file\_length()**

**2-2. 파일시스템 파일관리 close 추가 구현**

**스레드가 종료될 때 종료되는 스레드가 열었던 파일 목록을 모두 닫아준다. 이를 위해 userprog/process.c의 process\_exit() 함수에서 현재 스레드의 files 배열을 돌며 모든 파일에 대해 file\_close를 실행한다.**

1. **파일 쓰기 금지 (Critical Section 문제 해결)**

**여러 스레드가 같은 파일을 여는 것은 허용된다. 그리고 동시에 읽는 것도 가능하다. 하지만 다른 스레드가 파일을 읽는 동안에 또다른 스레드가 같은 파일을 편집할 수는 없다. 이를 위해 filesys/file.c에서 제공하는 file\_deny\_write() 함수와 file\_allow\_write() 함수를 사용한다.**

**이론 상으로는 임의의 스레드가 파일을 읽는 file\_read() 함수를 부르기 전에 같은 파일에 대해 쓰기금지 file\_deny\_write() 함수를 부르고, file\_read()가 끝나고 나서 쓰기금지 해제 file\_allow\_write() 함수를 호출하도록 구현한 후, 파일 쓰기 함수인 file\_write() 함수를 호출할 때 deny\_write 값을 확인해서 쓰기 금지가 되어 있으면 쓰기 금지가 해제될 때까지 기다리도록 구현하면 된다. 이를 실제로 구현하기 위해 userprog/syscall.c에서 int read() 함수에 file\_read 전에 file\_deny\_write() 함수를 호출하도록 하였다. 그리고 userprog/syscall.c의 int write() 함수에서 deny\_write 여부를 체크한 후 대기를 semaphore로 구현하려고 하였으나 파일 쓰기와 일기가 일어나는 시점을 제대로 맞추지 못해서인지 여러 개의 파일을 읽기 및 편집하는 조건에서는 프로그램이 의도한 대로 동작하지 않았다.**

1. **스레드 간 execution 및 load 타이밍 맞추기**

**부모 스레드는 자식 스레드를 생성하기 위해 userprog/process.c의 process\_execute() 함수를 호출한다. Process\_execute() 함수는 threads/thread.c에 있는 thread\_create() 함수를 호출하여 새로운 자식 스레드를 생성한다. 이렇게 생성된 자식 스레드는 userprog/process.c의 start\_process() 함수를 호출하고, start\_process는 userprog/process.c의 load() 함수를 호출하여 자신이 수행해야 하는 프로그램 파일을 로드한다. 이 때, load() 함수는 자식 스레드가 수행할 파일을 정상적으로 로드했는지 파일 로드에 실패했는지 여부를 판단하는데, 부모 스레드의 process\_execute()가 자식의 load()보다 일찍 종료되어 부모는 자식이 정상적으로 프로그램을 실행했는지 여부를 알 수 없게 된다.**

**이 문제를 해결하기 위해 새로운 semaphore를 threads/thread.h의 struct thread에 추가하였다. 그리고 부모 스레드의 process\_execute 함수가 thread\_create 함수를 호출한 후에 spin lock을 걸어 execute가 종료되지 않도록 한 후에, 자식 스레드인 start\_process에서 load()가 끝난 후에 락을 해제하였다. 이렇게 하면 process\_execute는 자식 스레드의 로드 성공 여부를 확인한 후에 종료되어 process\_wait() 함수를 실행하고, 자식 스레드가 종료될 때까지 대기한다.**

**그런데 위와 같은 방법으로 프로그램을 구현하였더니 process\_execute에서 새롭게 생성된 자식 스레드에 스레드 아이디인 tid 멤버가 제대로 할당되지 않는 문제가 발생하였다. 이유를 알 수 없으나 새롭게 생성된 자식 스레드는 스레드 이름만 저장되고 tid가 제대로 저장되지 않아 process\_wait에서 자식 스레드로 인식하지 못해 프로그램이 곧바로 종료되었다. Process\_execute가 load를 기다리면 자식 스레드 아이디가 할당되지 않고, process\_execute가 load를 기다리지 않으면 스레드 아이디가 정상적으로 할당되는 이유를 찾지 못하여 실제 구현에 실패하였다.**

* 1. **연구원 역할 분담**
* **서인호: 파일시스템 관련 시스템콜 구현, 파일쓰기 금지 기능 구현, 부모 자식 스레드 간 싱크 문제 해결, 보고서 작성 전반**
* **팀원1: 보고서 마무리 및 검토**

1. **연구 결과**
   1. **합성 내용**

|  |
| --- |
|  |

|  |
| --- |
|  |

* 1. **제작 내용**
     1. **System call 파일시스템**

**파일 시스템에 관련된 시스템 콜은 userprog/syscall.c의 syscall\_handler 함수에 추가하였다. 이는 새로운 파일을 생성 및 삭제하고, 열고, 읽거나 쓰고, 닫는 등 파일관리에 대한 기능을 정의한다. 이와 관련된 기능은 모두 정상 동작함을 확인하였따.**

* + 1. **Deny\_write**

**여러 개의 스레드가 여러 개의 파일을 열고 동시에 편집하는 환경에서 쓰기금지 기능을 구현하는 데 실패하였다.**

* + 1. **Load & executable 문제**

**부모 스레드의 process\_execute가 자식 스레드의 load를 기다린 후에 종료되도록 구현하면 자식 스레드의 스레드 아이디가 정상적으로 생성되지 않는 문제가 발생하였다. 결론적으로 부모 스레드는 자식 스레드의 번호를 인지하지 못해 process\_wait가 정상 동작하지 않아 프로그램이 오동작하여 해당 기능 구현에 실패하였다.**

* 1. **시험 및 평가 내용**

|  |
| --- |
|  |

1. **기타**
   1. **연구 조원 기여도**
      1. **서인호 95%**
      2. **팀원1 5%**
   2. **소감**

**파일 관리가 유저 프로그램의 영역이라고 생각해 왔었는데, 파일 관리가 운영체제의 커널 영역이었음을 인지했다. 파일 읽기 및 쓰기 기능을 확장하면 프로그램을 열고 쓰는 일과 관련이 있다고 생각한다. 그런 의미에서 본 프로젝트 내용을 통해 운영체제가 사용자의 프로그램 접근 권한을 어떻게 관리하는지, 그리고 운영체제 레벨에서 발생할 수 있는 Critical Section 문제를 어떻게 해결할 수 있을지에 대해 고민해볼 수 있는 좋은 기회였다고 생각한다. 비록 원인을 알 수 없는 문제로 본 프로젝트의 문제를 모두 해결 및 구현하지는 못했지만, 이 문제에 대해 고민하고 시도해본 경험만으로도 운영체제의 역할 및 동작 원리에 대해 보다 깊게 이해할 수 있었다.**