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

담당 교수 / 분반 :박성용 / CSE4070-01

이름 / 학번 : 박성현 / 20181632

개발 기간 : 2022-10-07 ~ 2022-10-08

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

추가적으로 남은 system calls (create, remove, open, clos, filesize, read, write, seek, tell)을 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
       1. 이번 프로젝트에서 구현해야 하는 system call들은 file 관련된 함수들이다. 파일을 열고 닫으며 읽고 쓰고 하려면 file descriptor을 사용해야 한다. Fd[3]부터 빈칸에 새로운 파일이 들어왔을 때, 할당해준다.
    2. (이번 프로젝트에서 구현해야 하는) System Calls
       1. Create는 새로운 파일을 생성하고, remove는 파일을 제거한다. Open은 파일을 열고 close는 파일을 닫는다. Filesize는 파일의 크기를 반환하고 read는 파일로부터 읽어들인다. Write는 파일에 쓰고 seek는 파일의 커서위치를 옮긴다. 마지막으로 tell은 파일에서 현재 커서의 위치를 반환한다.
    3. Synchronization in Filesystem
       1. 이번 프로젝트에서는 세가지의 synchronization을 사용했다. 첫번째는 자식 process가 메모리에 load되기 전에 부모 process가 사라지는 걸 방지해주는 semaphore이고 하나는 여러 프로세스가 하나의 파일에 동시에 접근하지 못하게 하는 lock이다. 하나의 파일에 여러 process가 접근할 경우 원치 않는 반환 값을 얻을 수 있다. 마지막으로 파일에 write를 할 경우, 하나의 process만 접근할 수 있게 하는 deny\_write을 구현했다.

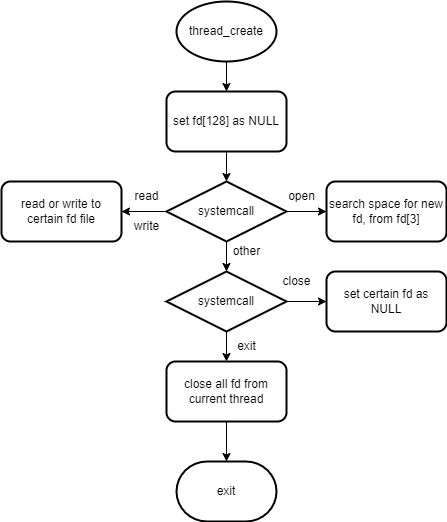
* 1. **개발 내용**
* 아래 항목의 내용만 서술
  + 1. File Descriptor: 구현에 이용할 자료구조와 선택한 이유를 서술
       1. 아주 간단한 배열 형식의 file descriptor를 사용했다. 명세서에 적혀 있듯이 128개의 file descriptor를 thread 구조체에 추가하고 초기화 시켰다. 새로운 file을 열 때, 0,1,2는 이미 stdin, out, error로 할당이 되어 있기에 3부터 빈칸을 찾아 file descriptor를 할당해 주었다. 또한, close시에 현재 사용중인 file descriptor을 NULL로 초기화 시킨다. 마지막으로 exit시에 현 process의 thread에 대해 열려 있는 모든 file descriptor를 close해주는 작업도 추가했다.
    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)
       1. Exit: 종료하기 전, 열려 있는 file descriptor를 닫아주는 기능 추가.
       2. Exec: locking 추가
       3. Read: fd가 2보다 클 때, 즉 stdin이 아닐 때, 입력 받은 fd로부터 file\_read함수를 사용해서 파일을 읽고 읽은 byte수를 반환하는 기능 추가.
       4. Write: read와 동일하게 stdout이 아닐 때, 들어온 fd로 값을 쓰는 기능 추가. 또한 file\_deny\_write를 통해 file에 쓸 수 있는 권한이 생길 때까지 기다리는 기능 추가.
       5. Create: 인자로 받은 이름과 크기의 새로운 파일을 생성하고 성공 여부를 반환한다.
       6. Remove: 인자로 받은 파일이름의 파일을 제거하고 성공 여부를 반환한다.
       7. Open: 인자로 받은 파일이름의 파일을 연다. File descriptor 3번부터 빈 칸을 찾아 새로운 파일을 할당하고 할당된 file descriptor 번호를 반환한다. 또한, 이미 열려 있는 파일이라면 그 파일에 대한 접근이 끝날 때까지 기다린다.
       8. Filesize: 인자로 받은 fd에 할당된 파일의 크기를 반환한다.
       9. Seek : 인자로 받은 fd에 할당된 파일의 커서를 position으로 옮긴다.
       10. Tell: 인자로 받은 fd에 할당된 파일의 현재 커서위치를 반환한다.
       11. Close: 인자로 받은 fd에 할당된 파일을 닫는다. 파일을 닫고 fd에 NULL값을 할당해준다.
    3. Synchronization in Filesystem: Lock, Semaphore를 어떻게 이용할 수 있는지 각각에 대해 설명 (다른 방법을 서술해도 되지만 lock과 semaphore는 반드시 포함해야 함)
       1. 가장 먼저 사용한 synchronization은 여러 process가 하나의 파일에 쓰기작업을 동시에 하지 못하게 하는 것이다. File\_deny\_write 함수를 사용하여 만약 해당 파일이 deny\_write상태라면 다른 process나 file descriptor에 열려있다는 의미이므로 닫힐 때까지 기다린 후 쓰기 작업을 실행 할 수 있도록 한다. 두번째는 file lock이다. 내장된 lock함수를 사용해 파일에 접근할 수 있는 process를 하나로 제한한다. 파일에 접근하는 모든 system call 함수, 즉 이번 프로젝트에서 구현한 함수와 exec함수의 처음과 끝에 lock\_acquire과 lock\_release를 통해 접근할 수 있는 process를 제한한다. 마지막은 부모와 자식 간의 semaphore이다. 프로세스가 생성되면 load함수를 통해 메모리에 올라가게 된다. 하지만 부모 프로세스가 자식 프로세스가 생성되어 메모리에 올라가기 전에 죽어버린다면 자식은 orphan process가 되게 된다. 이 경우를 방지하기 위해 자식 process가 load되고 메모리에 올라가기 전까지 부모 process에 semaphore을 통해 lock을 잡아준다.

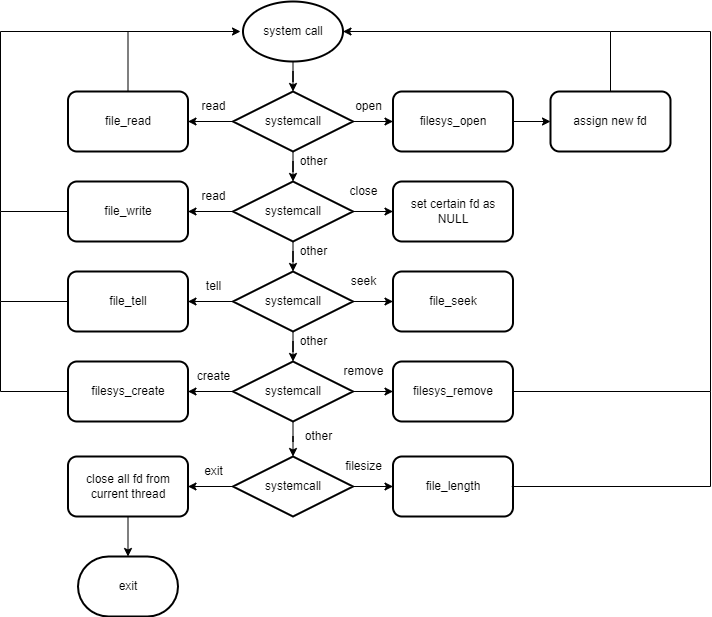
1. **추진 일정 및 개발 방법**
   1. **추진 일정**

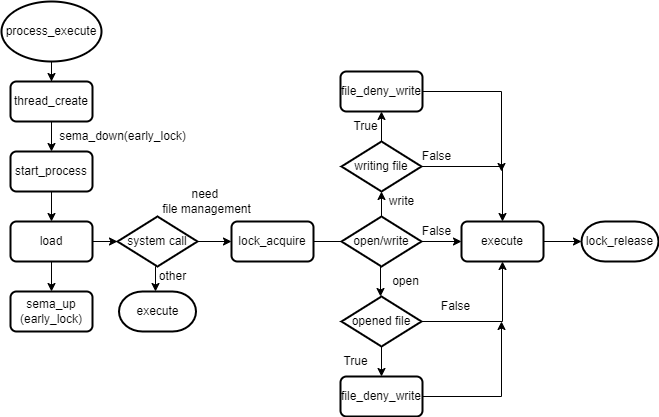
* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성
  + 10-07: 내장되어 있는 API를 호출하여 구현할 수 있는 system call 작성
  + 10-08: synchronization 구현
  1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드
    - Syscall.c
      * exit: 종료 전, 값이 할당되어 있는 fd 초기화
      * exec: file lock 추가
      * write: fd 값이 3 이상일때, 파일에 쓰고, 파일쓰기 권한을 확인하는 작업 추가
      * read: fd 값이 3 이상일때, 파일에서 읽는 작업 추가
    - process.c
      * process\_execute
        + 자식 프로세스가 load 될 때까지 sema\_down.
        + 종료된 자식을 list에서 제거 – process\_wait으로 tid를 넘겨준다.
      * Start\_process
        + Load 함수를 호출 한 후, sema\_up을 통해 lock 해제.
    - Thread.h
      * Thread structure에 자식 프로세스 load를 기다리는 semaphore early\_lock추가. 부모 프로세스를 가르키는 struct thread \*parents 추가. File descriptor struct file \*fd[128]추가
    - Thread.c
      * Init\_thread에서 parents를 새로 지정. Early\_lock 시작시키는 작업 추가, thread의 fd NULL로 초기화
  + 수정하거나 추가해야 하는 자료구조
    - Process\_execute에서 강제 종료되서 exit\_stat이 -1인 child 리스트에서 제거
    - Syscall.c에서 파일에 write시에 그 파일이 deny\_write면 file\_deny\_write함수를 통해 접근중인 프로세스나 fd가 없을 때까지 기다린 후 write작업 수행
    - open시에도 현재 thread 이름과 open하는 파일의 이름이 동일하면 그 파일에 대한 접근이 사라질 때까지 file\_deny\_write함수를 통해 기다림.
  + 수정하거나 추가해야 하는 함수
    - 추가로 구현해야 하는 system call함수 이외에 추가한 함수는 없다.

1. **연구 결과**
   1. **Flow Chart**

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

****

* + ****System call
  + Synchronization in file system

****

* 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**
  + Struct file \*fd[128]을 thread 구조체에 포함시키고 init\_thread시 모든 fd를 NULL로 초기화 시켰다. Open시에, fd[3]부터 돌면서 빈 index를 찾아 새로 연 파일을 할당해주었다. 0,1,2는 stdin, stdout, stderror이 할당되어 있기 때문에 3부터 시작했다. Read, write할 때, fd가 2보다 크면 파일에서 읽거나 쓸 수 있도록 구현했다. 또한, exit시에 current thread의 모든 fd를 close해주는 기능도 추가했다. close함수 구현 시, 같은 fd를 두번 닫는 경우를 대비해 close하는 fd에 대해 NULL을 대입했다.
* **System call**
  + Exit: thread\_exit을 호출하기 전, thread의 모든 fd에 close하는 작업을 추가했다. 파일을 닫지 않은 채로 두면 다른 process가 접근할 수 없기 때문이다.
  + Read: fd가 2보다 크면 파일에서 읽는다는 의미이기에 else if를 추가해 조건을 추가했다. 이 경우, 받은 fd에 값이 들어있지 않으면 exit(-1), 값이 있다면 파일을 file\_read함수로 읽고 읽은 byte만큼 반환한다.
  + Write: read와 마찬가지로 fd가 2보다 클때의 조건을 추가했다. write역시 받은 fd에 값이 없으면 exit(-1)을 한다. 만약 fd에 있는 파일에 누군가 접근 중이라면 접근이 종료될 때까지 기다렸다가 접근할 수 있게 한다. 그 후, file\_write함수를 사용해 파일에 직접 쓰고 쓴 byte만큼 반환한다.
  + Create: 받은 파일명으로 받은 크기만큼의 파일을 filesys\_create 함수를 통해 생성한다. 파일명이 NULL일경우, exit(-1)을 호출한다. 성공 여부를 bool 타입으로 반환한다.
  + Remove: 받은 파일명으로 된 파일을 filesys\_remove를 통해 제거한다. Create와 같이 NULL filename이면 exit(-1)을 호출하고 반환 값은 성공여부이다.
  + Open: 받은 파일명으로 된 파일을 연다. 파일명이 없으면 exit(-1)을 호출한다. 그 파일의 주소가 user인지 확인 후, filesys\_open함수를 사용해 f라는 file structure에 할당한다. 그 파일이 존재하지 않으면 -1을 반환한다. Fd[3]부터 돌아가며 NULL이 할당되어있는 index를 찾는다. NULL이면 아직 아무것도 할당되지 않은 fd이므로 그 index에 파일 f를 할당한다. 할당 전, 현 thread의 이름과 파일명이 겹치면 파일이 열려있다는 의미이므로 파일에 대한 접근이 끝날 때 까지 기다린다. 성공 시, 할당된 fd를 int형으로 반환한다.
  + Filesize: 받은 fd에 파일이 없으면 exit(-1)을 호출한다. File\_length함수를 사용해 fd에 할당된 파일의 크기를 반환한다.
  + Seek: 받은 fd에 파일이 없으면 exit(-1)을 호출한다. File\_seek함수를 사용해 받은 인자 position으로 fd의 커서를 움직인다. 즉, 다음에 읽을 위치를 position으로 옮기는 기능이다.
  + Tell: 받은 fd에 파일이 없으면 exit(-1)을 호출한다. File\_tell함수를 사용해 fd에 할당된 파일의 커서 위치를 반환한다.
  + Close: 받은 fd에 파일이 없으면 exit(-1)을 호출한다. File\_close함수를 사용해 fd에 할당된 파일을 닫는다. File\_close에는 닫은 fd를 초기화 시켜주는 작업이 포함되어 있지 않으므로 파일을 닫고 난 후, fd를 NULL로 초기화 시켜준다.
* **Synchronization in file system**
  + 한 파일에 대해 하나의 process만 접근해 쓸 수 있다. 그러므로 그 파일에 대한 fd의 deny\_write를 확인해줘야 한다. 가장 먼저, file open시에 strtmp함수를 사용해 현 thread의 이름과 파일명을 검사한다. 같으면 현재 thread에서 그 파일을 사용하고 있다는 의미이므로 file\_deny\_write를 사용해 그 파일에 대한 접근이 끝날때까지 기다린다. 그 후, fd에 파일을 할당해준다. 또한, write시에 쓰려는 파일에 대해 deny\_write를 체크한다. 만약 set 되어있으면 file\_deny\_write를 통해 파일에 대한 접근이 끝날때까지 기다린다. 그 후, write를 수행한다.
  + File에 대한 작업을 수행할 때, 여러 프로세스가 접근하게 되면 예상치 못한 반환 값을 얻을 수 있다. 그러므로 lock을 걸어줘야 한다. Syscall\_init에 lock\_init을 통해 lock을 실행시킨다. Lock이 필요한 file에 접근하는 모든 system call, 즉 exec, create, open, remove, filesize, close, seek, tell에 대해 system call API를 호출하기전에 lock\_acquire로 lock을 걸어주고 return 하기전에 lock\_release로 풀어준다. Lock은 이미 걸려있는 상태에서 다시 걸게되면 kernel panic이 발생하기에 lock\_release로 풀어주는 작업이 매우 중요하다.
  + 마지막으로 부모와 자식 간의 semaphore이다. 모든 process는 생성된 후, load를 통해 메모리에 올라간다. 하지만 load되기전, 부모 process가 사라지면 orphan process가 되기 때문에 자식 precess가 load될 때 까지 부모 process에 lock을 걸어줘야 한다. Thread structure에 새로운 semaphore early\_lock과 structure \*thread parents를 생성 후, early\_lock을 init\_thread시에 0으로 생성한다. 그리고 이미 돌아가고 있는 thread를 현재 thread의 parents로 지정한다. Process execute시 thread\_create로 새로운 thread를 생성하는데 이 작업이 실행된 후, sema\_down을 통해 lock을 걸어준다. start\_process에서 load가 완료된 후, sema\_up을 통해 현재 thread의 parents의 lock을 풀어준다. Sema\_up 호출시 current\_thread가 아닌 current\_thread->parents의 lock을 풀어야 한다. Start\_process는 자식 thread에서 수행되는 함수이기 때문이다. 또한, 종료된 자식 process를 list에서 제거하는 작업도 수행해야 한다. Process \_execute의 마지막에 list를 돌면서 exit\_stat이 -1인 thread, 즉 강제로 종료된 thread는 process\_wait을 호출해 제거하는 코드를 추가했다.
  + Lock을 read와 write에 걸었을 시, read-bad-ptr, write-bad-ptr이 정상정으로 수행되지 않았다. Read와 write에서는 lock에 대한 작업을 수행하지 않는다.
  1. **시험 및 평가 내용**
* make check 수행 결과를 캡처하여 첨부
* 