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

담당 교수 / 분반 : 김영재 교수 / 4

이름 / 학번 : 이재영 / 20171669

개발 기간 : 2021.10.05 ~ 2021.10.12

1. **개발 목표**

Base File System(기본 파일 시스템)을 바탕으로 file system과 연관이 있는 system call을 구현하고 file system의 synchronization을 구현한다.

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

최대 131의 size의 파일 구조체 포인터 file descriptor를 선언한다. File descriptor가 필요한 이유는 file system 기능을 원활하게 수행하도록 하기 위함이다. 간단히 open을 하면 file descriptor가 비어 있는 곳에 파일을 넣는데 여러 file descriptor가 연결될 수도 있기 때문에 배열로서 구현한다.

* + (이번 프로젝트에서 구현해야 하는) System Calls

Create/remove/open/close/seek/tell/filesize를 새로 구현하고 read, write를 프로젝트 1번에서 단순히 구현했던 것을 추가로 구현한다.

* + Synchronization in Filesystem

기본 Base file system의 경우는 internal synchronization이 구현되어 있지 않으므로 새로 lock 구조체를 만들어 lock\_acquire과 lock\_release를 사용해 file system synchronization을 구현한다. 또한 process가 제대로 load, exit 등의 과정을 수행할 수 있도록 semaphore를 사용해서 process사이에서의 synchronization도 가능하게 한다.

* 1. **개발 내용**

1. File Descriptor

File Descriptor를 구현하기 위해 src/threads/thread.h에서 파일 구조체 포인터인 struct file\* file\_des[131]을 추가하고 thread.c의 init\_thread에서 131번을 돌면서 모든 file\_des을 NULL로 초기화했다. 여기서 131인 경우 file descriptor은 최대 128개의 size를 가질 수 있는데 이미 file descriptor 0, 1, 2는 reserved되어 있으므로 128 + 3 = 131개의 파일 구조체 포인터 배열을 선언했다. File descriptor는 연결이 안 되어있는 즉, NULL인 곳을 찾아 open한 file를 연결할 수 있다. 여러 파일이 연결될 수 있기 때문에 배열로 구현했고 system call exit을 호출할 때마다 즉, thread를 exit할 때마다 file descriptor가 연결되어있는 경우 close system call를 호출해 연결되어 있는 file들을 끊은 뒤 값을 NULL로 초기화 하는 과정을 거친다.

2. System Calls

Open - 입력 받은 file의 이름을 가진 파일을 열어주는 system call이다. 파일을 열어주고 file descriptor에 연 파일을 저장한다. src/filesys/filesys.h에 있는 filesys\_open함수를 이용한다.

Create - 입력 받은 명령어의 이름을 가진 파일을 size만큼 할당해주는 system call로 src/filesys/filesys.h에 있는 filesys\_create함수를 이용한다.

Remove – 입력 받은 파일을 삭제해주는 함수로 src/filesys/filesys.h에 있는 filesys\_remove함수를 이용한다.

Close – 입력 받은 fd를 가지는 파일을 닫아주는 함수로 src/filesys/file.h에 있는 file\_close함수를 이용한다.

Read – 프로젝트 1에서 STDOUT을 구현했고 이번 프로젝트에서는 file을 읽는 경우도 추가해준다. src/filesys/file.h에 있는 file\_read함수를 이용한다.

Write – 프로젝트 1에서 STDIN을 구현했고 이번 프로젝트에서는 file에 write를 수행하는 경우를 추가해준다. src/filesys/file.h에 있는 file\_write함수를 이용한다.

Filesize – fd에 맞는 파일의 size를 반환하는 system call로 src/filesys/file.h에 있는 file\_length함수를 이용한다.

Seek – fd에 해당하는 파일의 위치를 입력 받은 loc값으로 만드는 기능을 한다. src/filesys/file.h에 있는 file\_seek함수를 이용한다.

Tell – fd에 해당하는 파일의 위치를 알려주는 system call로 src/filesys/file.h에 있는 file\_tell함수를 이용한다.

3. Synchronization in Filesystem: Lock, Semaphore

Synchronization은 동기화 방법으로 semaphore를 사용한다. 프로젝트2에서 사용하는 lock 방법 또한 semaphore의 일종으로 lock구조체 안에 semaphore가 element로 있으며 lock을 사용하는 lock\_acquire, lock\_release 함수 또한 함수 안에 semaphore를 사용하는 sema\_up, sema\_down을 수행한다. 따라서 Semaphore에 대한 설명을 하고 lock을 사용하는 것에 대해서는 4-B에서 자세히 설명한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Semaphore는 Critical Section에 여러 thread가 접근하는 것을 막아주는 기법이다. Critical Section이란 여러 thread가 공유 데이터를 가지고 수행할 때, 각 thread(process)에서 공유 데이터를 접근하는 code부분을 말하며 semaphore은 이를 방지해준다.

이번 프로젝트에서는 file system을 사용하는데 Base file system에서는 internal synchronization이 구현되어 있지 않으므로 file에 대한 system call을 수행하면서 여러 thread가 동시에 접근하는 것을 방지하지 못하는 문제가 발생한다. 이를 위해서 lock 구조체를 사용해서 file과 관련된 명령을 수행하기 전에 lock\_acquire로 다른 thread가 file에 접근하는 것을 방지하고 수행이 끝나면 lock\_release를 호출해서 다시 접근이 가능하도록 하면서 synchronization을 구현할 수 있다. Semaphore의 경우에는 parent process가 child process가 load되기도 전에 끝나거나 하는 등의 문제를 막기위해 wait, execution에서 sema\_up, sema\_down을 사용해 문제를 해결할 수 있다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

10/5 – 10/7 : Base File System에 대해 이해한 후 파일 descriptor와 관련된 구조체와 system call를 구현

10/8 – 10/9 : Synchronization을 구현하기 위해 lock과 semaphore에 대해 조사한 후 pintos에 구현

10/10 : multi-oom문제의 원인인 메모리 누수의 문제가 있는 코드를 수정

10/11 – 10/12 : 보고서 작성

* 1. **개발 방법**

<src/userprog/syscall.c>

* + Lock 구조체를 추가하고 syscall\_init함수에서 lock\_init함수를 통해 lock을 초기화해준다.
  + Read, Write System call인 경우에 대해서 lock\_acquire, lock\_release함수를 사용해서 synchronization을 수행하고 fd가 각각 STDOUT, STDIN의 경우가 아닌 fd가 3인 경우에 대한 read, write기능을 추가한다.
  + Exit system call인 경우 file descriptor에 연결되어 있는 파일이 열려 있는 경우에 대해서 닫아주는 코드를 추가한다.
  + 새로 만드는 system call(create, remove, close, open, seek, tell, filesize)의 기능을 수행하는 함수를 새로 추가한다. 이 때, file system을 사용하는 경우 lock\_acquire, lock\_release함수를 사용해서 synchronization을 수행한다.

<src/userprog/process.c>

* + Process\_execute함수에서 child가 load가 되기도 전에 parent process가 종료되는 것을 semaphore를 통해서 막는다. 제대로 load가 되지 않은 child가 있는 경우에 대해서 wait시켜주는 반복문을 추가해준다.
  + Start\_process함수에서 현재 thread의 parent를 semaphore를 사용해 sema\_up을 해준다. 여기서 parent는 src/threads/thread.c에서 running\_thread()로 설정되어 있다. 여기서 load를 정상적으로 하지 못했다면 현재 thread의 check\_child를 1로 설정하고 syscall.c의 exit system call를 호출한다.

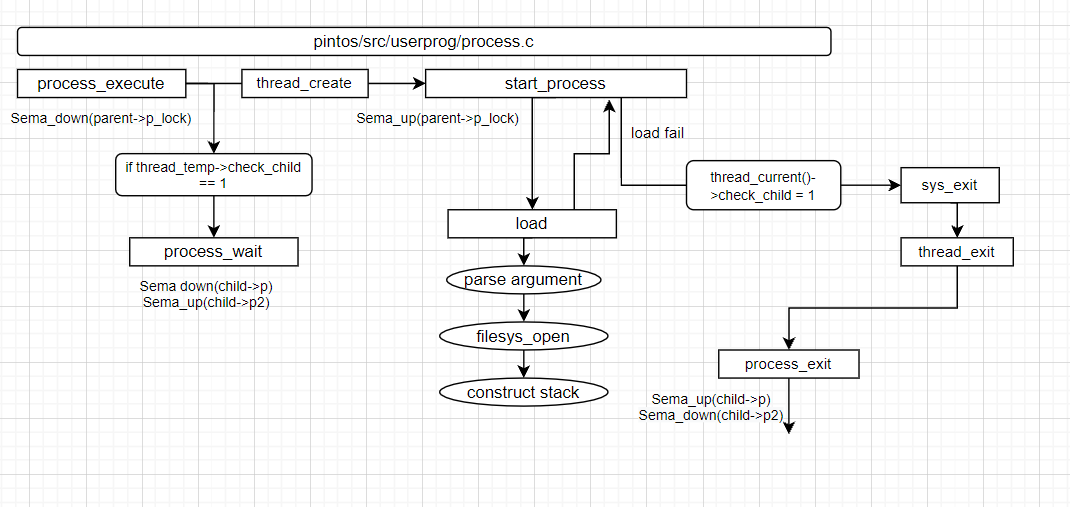
<src/threads/thread.h>

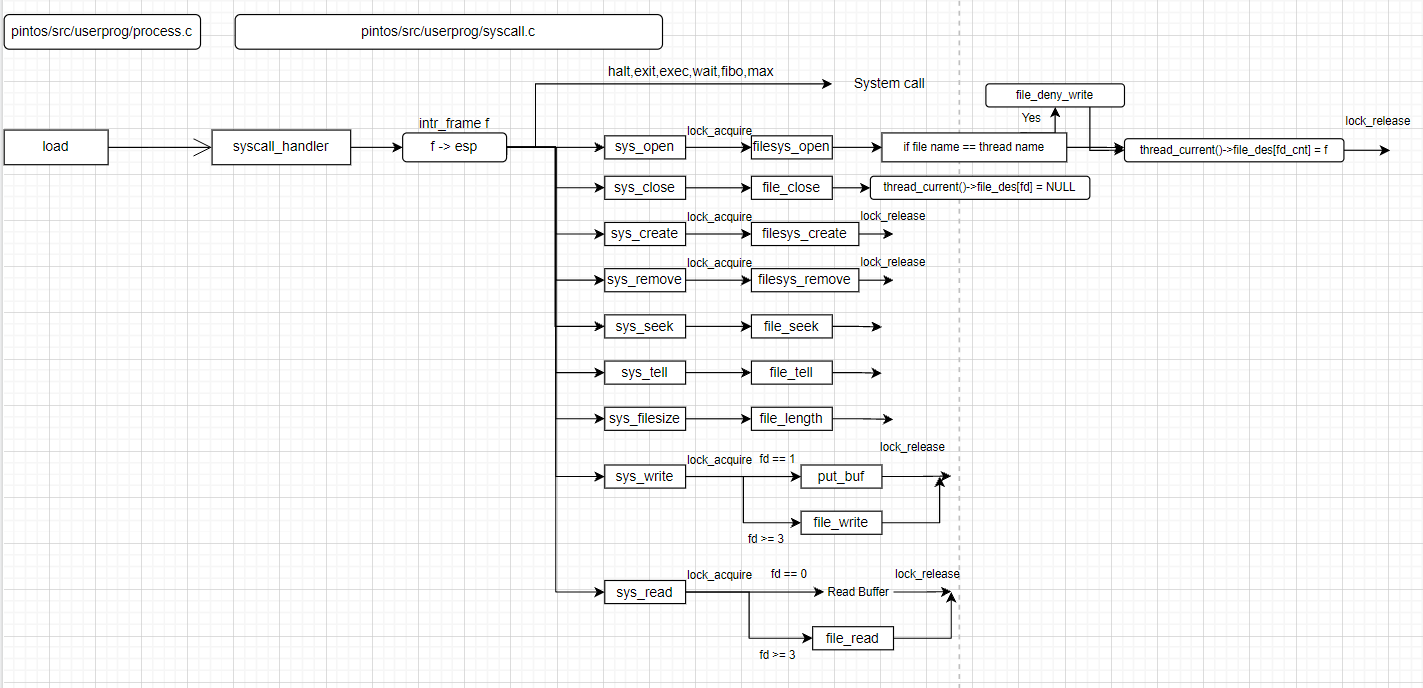
* + Thread 구조체에 새로 semaphore p\_lock를 선언하고 integer type의 check\_child 값을 선언하고 thread 구조체 포인터 parent, file 구조체 포인터 배열 file\_des[131]를 새로 추가한다.

<src/threads/thread.c>

* + Thread.h에서 선언한 semaphore p\_lock, check\_child를 0으로 초기화하고 file\_des 배열을 size 131만큼 반복문을 통해 NULL값으로 초기화했다. Parent의 경우는 running\_thread()로 설정했다. Running thread()는 현재 running thread를 return해주는 함수이다.

1. **연구 결과**
   1. **Flow Chart**
      1. File descriptor/System Call





* 1. **제작 내용**

<Src/threads/thread.c(thread.h)>

**텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**

새로 선언한 변수들과 init\_thread에서 초기화한 코드이다.

1. File Descriptor

Src/threads/thread.h에서 다음과 같이 file descriptor를 선언하고 thread.c의 init\_thread에서 배열의 모든 값을 NULL로 초기화했다. :

텍스트이(가) 표시된 사진

자동 생성된 설명 또한 file descriptor의 경우 file system을 사용하는 모든 system call에서 사용하기 때문에 이에 대해서는 아래의 System call 구현에서 추가로 설명하겠다.

2. System Call(Create, Remove, Open, Close, Read, Write, Seek, Tell, Filesize)

* Create

**텍스트이(가) 표시된 사진

자동 생성된 설명**

Create system call의 경우 sys\_create이라는 bool type함수를 생성해서 반환값을 가지고 f->eax에 넣는 작업을 수행했다.

Create system call은 입력 받은 파일의 이름을 size만큼 생성해주는 기능을 하며 src/filesys/filesys.h에 정의되어 있는 filesys\_create 함수를 사용하여 구현했다. 파일의 이름이 NULL인 경우 sys\_exit(-1)를 호출했으며 filesys\_create 함수를 사용할 때, lock\_acquire와 lock\_release함수를 사용해 synchronization을 구현했다.

* + Remove

텍스트이(가) 표시된 사진

자동 생성된 설명

Remove system call의 경우 sys\_remove라는 bool type함수를 생성해서 반환값을 가지고 f->eax에 넣는 작업을 수행했다.

Remove system call은 입력 받은 파일을 삭제해주는 기능을 해준다.

* + Open

텍스트이(가) 표시된 사진

자동 생성된 설명

open system call의 경우 sys\_open이라는 integer type함수를 생성해서 반환값을 가지고 f->eax에 넣는 작업을 수행했다.

open system call은 입력 받은 파일을 열어주는 기능을 하며 src/filesys/filesys.h에 정의되어 있는 filesys\_open 함수를 이용했다. Parameter로 받은 file\_name이 NULL인 경우 -1을 return했다. 그 후, lock\_acquire함수를 호출했고 return 값을 반환하는 모든 경우에 대해, lock\_release를 호출하면서 synchronization을 구현하면서 파일을 열고 있는 도중 수정/삭제 등의 접근을 방지했다. File\_sysopen함수를 호출해서 NULL이 아닌 경우 thread구조체에서 설정한 file descriptor의 크기(131)만큼 while문을 반복하면서 file descriptor가 NULL인 경우 file\_sysopen을 통해 읽은 파일 구조체 포인터를 file descriptor에 넣고 file descriptor값을 return했다.

이 때, 현재 thread가 실행하고 있는 파일과 읽으려고 하는 파일의 이름이 같은 경우, file\_deny\_write함수를 호출해서 해당 파일에 새로 write하는 것을 방지했다.

* + Close

텍스트이(가) 표시된 사진

자동 생성된 설명

Close system call은 void type의 sys\_close함수를 호출해 수행했다.

Close는 입력 받은 file descriptor를 가지는 파일을 닫아주는 system call로 현재 thread의 file descriptor를 임시 파일 구조체 포인터 temp를 선언해서 저장해준 후 src/filesys/filesys.h의 file\_close함수에 temp를 보내면서 호출한 후, 현재 thread의 file descriptor값을 NULL로 초기화해주었다. 혹시라도 file descriptor가 NULL인 경우를 대비해 NULL인 경우 함수 초반부에서 sys\_exit(-1)를 호출해주었다.

* + Read

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

fd가 0인 경우는 STDOUT으로 문자를 읽는 기능을 하고 f->eax에 읽은 size만큼을 넣는다. (프로젝트1 구현 내용)

Read를 수행하기 전에 lock\_acquire를 호출했다. 다음 코드는 fd가 3이상인 경우에 대한 내용인데 이는 file을 읽는 경우에 대한 내용으로 현재 thread의 file descriptor가 NULL인 경우 lock\_release를 호출하고 exit system call을 수행한다. 정상적인 read system call인 경우는 src/filesys/file.h에 선언된 file\_read함수에 현재 thread의 fd값과 buffer, size에 해당하는 (void\*)\*(uint32\_t\*)(f->esp+8), (unsigned)\*((uint32\_t\*)(f->esp+12))를 인자로 호출해 반환 받은 값을 f->eax에 넣었다. 모든 기능을 끝내면 lock\_release함수를 호출하면서 synchronization을 구현했다.

* + Write

텍스트이(가) 표시된 사진

자동 생성된 설명

fd가 1인 경우는 STDIN으로 문자를 쓰는 기능을 하고 f->eax에 size((unsigned)\*((uint32\_t\*)(f->esp+12)))만큼을 넣는다. (프로젝트1 구현 내용)

Write를 수행하기 전에 lock\_acquire를 호출했다. 다음 코드는 fd가 3이상인 경우에 대한 내용인데 이는 file에 write를 수행하는 경우에 대한 내용으로 현재 thread의 file descriptor가 NULL인 경우 lock\_release를 호출하고 exit system call을 수행한다. 정상적인 write system call인 경우는 src/filesys/file.h에 선언된 file\_write함수에 현재 thread의 fd값과 buffer, size에 해당하는 (void\*)\*(uint32\_t\*)(f->esp+8), (unsigned)\*((uint32\_t\*)(f->esp+12))를 인자로 호출해 반환 받은 값을 f->eax에 넣었다. 모든 기능을 끝내면 lock\_release함수를 호출하면서 synchronization을 구현했다.

* + Seek

텍스트이(가) 표시된 사진

자동 생성된 설명

seek system call은 void type의 sys\_seek함수를 호출해 수행했다.

Seek system call은 parameter로 넘어온 fd에 해당하는 파일의 위치를 parameter로 넘어온 loc으로 설정하는 기능으로 src/filesys/filesys.h의 file\_seek함수를 사용했다. 현재 thread의 fd값이 NULL이라면 sys\_exit(-1)를 호출했다.

* + Tell

텍스트이(가) 표시된 사진

자동 생성된 설명

Tell system call은 unsigned type의 sys\_tell 함수를 호출해 수행했다.

Tell system call은 parameter로 넘어온 fd에 해당하는 파일의 위치를 알려주는 기능을 하며 src/filesys/filesys.h의 file\_tell함수를 사용했다. 현재 thread의 fd값이 NULL이라면 sys\_exit(-1)를 호출했다.

* Filesize

텍스트이(가) 표시된 사진

자동 생성된 설명

filesize system call은 integer type의 sys\_filesize함수를 호출해 수행했다.

filesize system call은 parameter로 넘어온 fd에 해당하는 파일의 크기를 알려주는 기능으로 src/filesys/filesys.h의 file\_length함수를 사용했다. 현재 thread의 fd값이 NULL이라면 sys\_exit(-1)를 호출했다.

3. Synchronization in Filesystem

텍스트이(가) 표시된 사진

자동 생성된 설명

File system과 관련된 함수를 수행할 때 lock 구조체를 사용한다. Lock 구조체는 src/userprog/syscall.c에 새로 정의하고 함수 syscall\_init에서 lock\_init 함수로 선언한 lock 구조체를 initialize했다. 각각의 system call에서 lock을 사용하는 경우에 대해 lock\_acquire, lock\_release를 호출한 것을 위의 system call 설명에서 확인할 수 있다.

<src/userprog/process.c>

텍스트이(가) 표시된 사진

자동 생성된 설명

process\_execute함수에서 새로 추가한 코드이다. Thread를 create해서 child thread를 생성하고 child thread가 정상적으로 수행될 때까지 parent thread를 semaphore로 lock시킨다. 그렇지 않으면 child thread가 load되기 전에 parent thread가 빨리 끝나는 문제가 발생한다. 이는 semaphore p\_lock을 sema\_down시키는 방식으로 해결했다. Child thread를 돌면서 thread 중에 check\_child의 값이 1인 경우에는 process를 wait한다. process\_wait 함수를 호출해서 해당 thread의 parent의 child list에서 제거한다. (이 과정은 프로젝트1의 process\_wait함수 구현과정에서 구현했다) check\_child는 load가 성공적으로 되지 않은 경우의 상황에 대한 integer 변수로 load가 제대로 되지 않은 thread인 경우 1로 설정했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위의 코드는 start\_process 함수에서 추가한 코드로 현재 thread의 parent의 semaphore p\_lock을 sema\_up을 해준다. 이 p\_lock은 process\_execute에서 sema\_down이 된다. Load가 제대로 되지 않은 경우는 현재 thread의 check\_child를 1로 설정해서 process\_execute함수에서 check\_child를 확인해서 제대로 load되지 않은 것을 확인하고 process\_wait를 수행할 수 있도록 해준다. 그 후에 src/userprog/syscall.c에서 선언한 sys\_exit(-1)를 호출한다.

<src/userprog/exception.c>

텍스트이(가) 표시된 사진

자동 생성된 설명

kernel영역을 접근하는 경우의 not\_present 가 0이 아닌 경우를 추가하여 예외사항 처리를 수행했다.

File system과 관련된 system call들의 경우 file descriptor를 확인하는 작업이 많기 때문에 현재 돌고 있는 thread의 file descriptor가 NULL인 경우 True와 아닌 경우 False를 반환해주는 함수 is\_fd\_NULL를 새로 구현했다.

개발 중에서 특히 multi-oom의 test에서 통과가 많이 되지 않았다. Src/test/no-vm에 있는 multi-oom.c와 명령어를 돌려보면서 문제가 메모리의 누수라는 것을 알아내서 여러 malloc으로 선언한 변수나 file descriptor가 NULL이 아니면 file\_close를 해주는 작업도 해주었는데 쉽사리 문제가 해결되지 않았다. 따라서 pintos의 처음 구현했던 부분부터 천천히 살펴봤더니 처음의 명령어를 parse하는 2차원 배열의 크기가 너무 크게 잡아서 error가 났다는 것을 확인할 수 있었다. 따라서 배열의 크기를 줄이니까 정상적으로 All 80 tests passed가 출력되었다.

* 1. **시험 및 평가 내용**

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명