OS Assignment#4

FUSE 기반 파일시스템 실습

학부 : 소프트웨어 학부

학번 : 20150291

이름 : 하현수

1. 소개

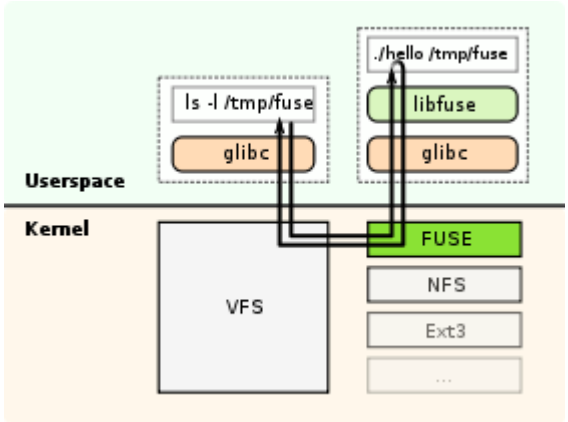
과제 개요

본 과제는 FUSE 기반 파일시스템을 이용하여 현재 실행 중인 프로세스 정보를 디렉터리 목록으로 보여주는 파일 시스템에서, rm 명령을 실행할 시, 해당 프로세스에 kill 명령을 전달하는 것을 목표로 합니다. pfs.c에 pfs_unlink()함수를 성공적으로 구현하였으며 이를 바탕으로 rm 명령을 호출할 경우 pfs_unlink()함수를 호출하여 해당 프로세스에 SIGKILL 명령을 보내 프로세스를 종료하게 하였습니다.
구현에 대한 테스트는 Vmware Workstation 14 버전을 통한 우분투 16.04 가상머신을 통해 진행하였습니다.
관련 연구는 FUSE에 대한 간략한 설명과, 본 코드에서 사용된 FUSE 라이브러리 함수 연구를 하였고, Proc 파일 시스템에 대해 조사하였습니다.

2. 관련 연구

FUSE(Filesystem in Userspace)

일반적으로 파일시스템은 Kernel Layer에서 개발해야하는데 개발을 위해서는 개발자가 Kernel에 대한 높은 이해도를 가지고 있어야 한다.
Kernel에 대해서 깊은 이해를 하고 있지 않더라도, 디버깅, 시스템 안정성 향상, 개발 속도 등을 위해서 filesystem을 자체적으로 구현하고 싶을 때 FUSE를 사용하는 경우가 많다.
간단하게 말해서, FUSE는 파일 시스템을 UserSpace 영역에서 개발할 수 있도록 도와주는 프레임워크이다.
Linux와 매킨토시의 경우에는 FUSE를 제공하고, 윈도우 기반 운영체제에서는 Dokan이 FUSE의 역할을 수행한다.
FUSE를 통해 UserSpace 영역에서 파일시스템 함수를 구현하고 그것을 FUSE 프레임워크에 넘기면, 내부적으로 Kernel Layer에서 해당 함수가 호출될 때, UserSpace 영역에서 구현된 함수를 다시 호출해준다.
사용방식도 매우 간단한데, function pointer를 인자로 넣은 fuse_main 함수를 호출하면 된다.



<PDF에 있는 참고 문헌 : Wikipedia 사진>
그림과 같이 Kernel 영역에서 Userspace의 libfuse를 호출하면서 사용자 영역에서 구현된 함수를 호출할 수 있는 것이다.

FUSE 라이브러리 함수

Stat, fstat 함수

stat()은 파일 경로명을 인자로 받고, fstat()은 파일 디스크립터를 인자로 받는다.
그 후 파일의 정보를 buffer 에 담게 된다.
int stat(const char * file_name, struct stat * buffer);
int fstat(int file_ds, struct stat * buffer); 이다.
stat 구조체는 파일에 대한 정보를 가지는 구조체로, 원형은 다음과 같다.

```
struct stat {
        dev_t     st_dev;      /* 파일 소유자의 디바이스 ID */
        ino_t     st_ino;      /* Inode number */
        mode_t    st_mode;     /* 파일 형식과 Mode */
        nlink_t   st_nlink;    /* 하드링크의 개수 */
        uid_t     st_uid;      /* 소유자의 User ID */
        gid_t     st_gid;      /* 소유자의 Group ID */
        dev_t     st_rdev;     /* Device ID (if special file) */
        off_t     st_size;     /* 파일 사이즈 */
        blksize_t st_blksize;  /* 파일 입출력 블록 사이즈 */
        blkcnt_t  st_blocks;   /* Number of 512B blocks allocated */
        struct timespec st_atim;  /* 최종 접근 시간 */
        struct timespec st_mtim;  /* 최종 변경 시간 */
        struct timespec st_ctim;  /* 최종 상태 변경 시간*/
    #define st_atime st_atim.tv_sec      /* Backward compatibility */
    #define st_mtime st_mtim.tv_sec
    #define st_ctime st_ctim.tv_sec
    };
```

Stat과 함께 쓰이는 에러는 ENOENT로, pathname에 아무것도 존재하지 않거나, pathname이 비어있는 문자열일 경우 반환되는 에러이다.

Getattr 함수

getattr(const char* path, struct stat* stbuf)의 원형을 가지는 함수이다.
stat 구조체에 해당 path의 파일 정보를 채워 넣는다. 파일시스템에서 사용 빈도가 높은 함수이다.

Readdir 함수

Caller에게 하나 혹은 그 이상의 Directory Entries를 반환한다. FUSE함수 중에서 가장 복잡한 함수로서, FUSE readdir 함수는 readdir(2) 시스템콜이나 readdir(3) 라이브러리 함수와는 비슷하지만 다른 함수이다. ls등의 명령어에 사용된다.
read와도 비슷하다. 우선 주어진 offset을 통해서 디렉토리를 찾고, 선택적으로 stat구조체에 getattr를 통해 정보를 채운다. 그 후에 filler function을 호출하여 buffer에 값을 입력받는다. 만약 이때 filler의 반환값이 0이 아니라면(정상적으로 수행되지 않았다면) 0을 반환한다.

unlink함수

unlink(const char* path)의 원형을 가지는 함수이다.
주어진 파일을 삭제하며 관련된 심볼릭 링크, 하드링크, 스페셜 노드 등을 삭제한다.
파일의 링크 카운트를 줄여주는 역할을 하며, 링크 카운트가 0 이 되면 그 파일은 삭제되었다고 판단할 수 있다.

과제에서 사용된 라이브러리는 위 함수들이 전부지만, 그 외에 자주 사용되는 라이브러리를 연구해보았다.

Link함수

link(const char* from, const char* to)의 원형을 가지는 함수이다.
from의 경로에서 to로 하드링크를 생성한다.

access함수

access(const char * path, mask)의 원형을 가지는 함수이다.
파일의 권한 정보를 불러올 때 사용된다.
만약 path가 존재하지 않으면 -ENOENT를 리턴하고, 만약 접근 권한이 없다면 -EACCESS를 반환한다.

mkdir함수

mkdir(const char* path, mode_t mode)의 원형을 가진다.
새로운 디렉토리를 만들 때 사용된다. 디렉터리의 권한 정보는 mode안에 인코딩 되어 들어간다.

rmdir함수

rmdir(const char * path)의 원형을 가진다.

비어 있는 디렉토리를 삭제할 때 사용된다.

chmod함수

chmod(const char * path, mode_t mode)의 원형을 가진다.

해당 path의 권한을 mode로 변경한다.

open함수

open(const char* path, struct fuse_file_info* fi)의 원형을 가진다.

파일을 열 때 사용되며, 경로의 존재 유무와 접근 권한 문제를 먼저 살펴야 한다.

read함수

read(const char * path, char * buf, size_t size, off_t offset, struct fuse_file_info* fi)의 원형을 가진다. Offset byte부터 Size 바이트 만큼을 path 파일에서 읽어 buf에 집어 넣는다.

Proc 파일 시스템 연구

유닉스에서 사용되는 Proc 파일 시스템은 운영체제의 각종 정보를 커널모드가 아닌 유저 모드에서 쉽게 접근할 수 있도록 만들어 줌으로서 시스템 정보에 일반 프로그래머가 쉽게 접근할 수 있도록 도와준다.

일반적인 파일 시스템은 상당한 오버헤드를 가지고 있다. 각 파일의 inode와 superblock 같은 객체를 관리해야 하며 이러한 정보를 필요할 때 마다 운영체제에 요청해야 하기 때문이다. 이로 인해 파일시스템의 데이터들은 서로 어긋날 수도 있으며, 단편화 현상이 발생할 수 있다. 운영체제는 이러한 모든 부분을 관리해 주어야 하기 때문에 당연히 상당한 오버헤드가 발생한다.

proc 파일 시스템은 이러한 일반 파일시스템의 문제점을 없애기 위해서 리눅스 커널에서 직접 파일 시스템을 관리하는 방법을 채택하였다. Proc 파일시스템은 커널 메모리에서 돌아가는 일종의 가상 파일 시스템이다. 메모리, 그것도 커널이 직접 관리를 하게 되어 당연히 빠른 속도를 자랑한다.

뿐만 아니라, /proc 은 커널메모리에서 유지하는 파일 시스템이므로, 별도의 장치(하드디스크)등이 필요하지 않다. 이러한 이유로 proc 파일 시스템은 임베디드 시스템 설계에 아주 중요한 요소로 쓰인다.

일반적인 파일 시스템 계층은 프로그래머를 위해서 POSIX 형식의 인터페이스를 제공한다. 이러한 일반 파일 시스템은 대용량의 데이터를 다루어야 하는 경우에는 매우 유용하지만, 고정적이고 처리해야 할 데이터의 양이 적은 분야에서는 오히려 비효율적이기 때문에 proc에서는 다루어야 할 정보가 이미 정해져 있고, 데이터 양이 그리 많지 않으므로 필요한 작업에만 최적화된 인터페이스를 제공한다.

3. 추가 기능 구현 방법

pfs_unlink 함수 구현 방법

```
static int pfs_unlink (const char *path)
{
  /* TODO: not yet implemented */
  char proc_path[4096];//get_proc 함수에 사용될 임시 버퍼
  char *path2;//임시 포인터 변수
  char *name;//파일의 이름을 담을 변수
  pid_t pid;//프로세스 아이디를 담을 변수
  path2 = strdup (path);
  name = basename (path2);
  if (get_proc (name, &pid, proc_path, sizeof (proc_path)) < 0)//프로세스 정보를 받아온다.
    {
      free (path2);
      return -ENOENT;//만약 get_proc에 실패했을 경우 -ENOENT 반환
    }
  kill(pid,SIGKILL);//unlink가 호출된 해당 프로세스 pid에 SIGKILL 신호 전달
  return 0;
}
```

Get_proc 함수를 통해서 해당 path의 프로세스를 불러온 후,

해당 pid에 kill을 통해서 SIGKILL시그널을 전달한다.

실행결과

```
root@server:~/assignment4-sample# ./pfs test/
root@server:~/assignment4-sample# sh &
[1] 2841
root@server:~/assignment4-sample# sh &
[2] 2888

[1]+  Stopped                 sh
```

먼저 프로세스(sh)를 2개 생성시킨 뒤, ls -l을 통해 test 폴더를 확인하면

```
rw-r--r-- 1 root             root             51584 Nov 20 16:03 2541-usr-bin-zeitgeist-daemon
rw-r--r-- 1 root             root             68896 Nov 20 16:03 2549-usr-lib-i386-linux-gnu-zeit
rw-r--r-- 1 www-data         www-data        228700 Nov 20 16:03 2690-usr-sbin-apache2
rw-r--r-- 1 www-data         www-data        228700 Nov 20 16:03 2691-usr-sbin-apache2
rw-r--r-- 1 root             root             14196 Nov 20 16:03 2774-usr-sbin-cupsd
rw-r--r-- 1 root             root             37432 Nov 20 16:03 2775-usr-sbin-cups-browsed
rw-r--r-- 1 lp               lp               11232 Nov 20 16:03 2780-usr-lib-cups-notifier-dbus
rw-r--r-- 1 root             root             29636 Nov 20 16:03 2837-.-pfs
rw-r--r-- 1 root             root              2368 Nov 20 16:04 2841-sh
rw-r--r-- 1 root             root              2368 Nov 20 16:04 2888-sh
rw-r--r-- 1 root             root              2368 Nov 20 16:04 2897-bin-sh
rw-r--r-- 1 root             root              2368 Nov 20 16:04 2901-bin-sh
rw-r--r-- 1 root             root             62860 Nov 20 16:04 2932-usr-bin-python3
rw-r--r-- 1 root             root              5440 Nov 20 16:04 2937-ls
```

2841-sh과 2888-sh를 확인할 수 있다.

(작업 환경이 VMware라서 알 수 없는 프로세스가 잔뜩 실행중인 모습)

그 후에 rm test/2841-sh 를 입력하면

```
[2]+  Stopped                 sh
root@server:~/assignment4-sample# rm test/2841-sh
rm: remove regular file 'test/2841-sh'? y
[1]-  Killed                  sh
```

Killed가 전달되며 sh이 종료된다.

만약 프로세스가 종료되지 않았다면 아무리 test 폴더에서 2841-sh을 삭제하더라도, 다시 새롭게 프로세스 정보를 불러오면서 다시 폴더가 생성되지만, 이런식으로 SIGKILL을 전달하면 프로세스가 종료되어 test 폴더 안에 2841-sh 파일이 사라져있을 것이다.

```
-rw-r--r-- 1 root             root             51584 Nov 20 16:03 2541-usr-bin
-rw-r--r-- 1 root             root             68896 Nov 20 16:03 2549-usr-lib
-rw-r--r-- 1 www-data         www-data        228700 Nov 20 16:03 2690-usr-sbi
-rw-r--r-- 1 www-data         www-data        228700 Nov 20 16:03 2691-usr-sbi
-rw-r--r-- 1 root             root             14196 Nov 20 16:03 2774-usr-sbi
-rw-r--r-- 1 root             root             37432 Nov 20 16:03 2775-usr-sbi
-rw-r--r-- 1 lp               lp               11232 Nov 20 16:03 2780-usr-lib
-rw-r--r-- 1 root             root             29636 Nov 20 16:03 2837-.-pfs
-rw-r--r-- 1 root             root              2368 Nov 20 16:04 2888-sh
-rw-r--r-- 1 root             root             87900 Nov 20 16:04 2941-usr-bin
-rw-r--r-- 1 root             root              7212 Nov 20 15:58 339-lib-syst
-rw-r--r-- 1 root             root             14856 Nov 20 15:58 365-lib-syst
-rw-r--r-- 1 root             root             21296 Nov 20 15:58 368-vmware-v
-rw-r--r-- 1 root             root             73312 Nov 20 16:04 3936-usr-lib
```

실제로 2841-sh 파일이 삭제된 것을 확인할 수 있다.

이상 보고서를 마칩니다. 감사합니다!