Operating System Report

Assignment 3

Professor	김태석 교수님
Department	Computer engineering
Student ID	2014722057
Name	김 진아
Class	화5 목6
Date	2016. 11. 25

◆ 3-1

A. Introduction

3-1과제는 하나의 프로세스에 대해 정보가 위치하는 가상 메모리 주소, 정보의 원본 파일의 전체 경로를 출력하는 시스템 콜을 설계하는 것이다. 이때 시스템 콜을 하나 생성한 뒤 여기에 wrapping한다. 시스템 콜의 이름은 __NR_ftrace이며 번호는 351번이다. 또한 시스템 콜 테이블에 sys_ftrace를 등록한다. 시스템 콜에서 수행할 작업은 return 0이다.

B. Conclusion

이번 과제를 구현하기 위해 실습자료를 많이 참고했다. 시스템 콜 351번 __NR_ftrace를 강의자료를 참고해서 생성했으며 wrapping하는 부분도 참고했다. 여기서 349번 __NR_add 대신 351 __NR_ftrace를 wrapping하도록 바꿔서 구현했다. __NR_ftrace는 프로세스의 정보를 출력하는 것이므로 ftrace함수는 프로세스 id로 반환해주며 프로세스 id를 인자로 받아야 한다. 또한 여기서 프로세스의 정보를 관리해주기 위해 struct mm_struct를 가리키는 포인트 변수를 선언해 task의 mm_struct를 가져오기 위해 get_task_mm함수를 사용했다. vm_area_struct list의 시작 주소를 가져오도록 했다. 이때 mmap함수를 사용해 호출한 프로세스의 가상 메모리 공산에 새로운 mapping을 생성했다. 가상 메모리에 있는 파일을 나타내기 위해 struct file 또한 선언했다. vm_area_struct list의 시작 주소가 메모리를 다 읽을 때까지 즉 가상메모리에 있는 정보를 다 읽을 때까지 while문을 돈다. 여기에서 메모리에 있는 파일을 읽어와 d_path함수를 통해 file의 절대경로를 문자열에 저장한다. 또한 이때 영역의 시작 주소와 끝 주소를 vm_start, vm_end 변수를 사용해 절대 경로와 함께 과제에 맞게 출력시킨다. 이 과정을 마치면 다음 영역으로 이동한다. 이런 식으로 프로세스의 가상메모리에 있는 모든 정보를 다 읽어온다. 이처럼 과제를 구현하기 전에 절대경로를 출력해야 하는데 무엇을 사용해야 되는지 몰랐다. 실습 강의자료에는 설명이 안되어 있어 고민하다가 다음과 같은 방법을 통해 d_path함수의 존재를 알게 되었다.

<Step 1> 절대 경로를 출력하는 함수를 찾는 것이므로 cscope을 이용해 path를 사용하는 파일들을 보기 위해 Find this C symbol 부분에 path를 입력했다.

```
C symbol: path

  File               Function                                  Line
0 dcache.h           <global>                                    14 struct path;
1 dcache.h           <global>                                   171 struct vfsmount *(*d_automount)(struct path *);
2 dcache.h           <global>                                   343 extern char *__d_path(const struct path *, const struct path *, char
                                                                    *, int );
3 dcache.h           <global>                                   344 extern char *d_absolute_path(const struct path *, char *, int );
4 dcache.h           <global>                                   345 extern char *d_path(const struct path *, char *, int );
5 dcache.h           <global>                                   346 extern char *d_path_with_unreachable(const struct path *, char *,
                                                                    int );
6 dcookies.h         <global>                                    19 struct path;
7 dcookies.h         <global>                                    47 int get_dcookie(struct path *path, unsigned long *cookie);
8 device-mapper.h    <global>                                   123 int dm_get_device(struct dm_target *ti, const char *path, fmode_t
                                                                    mode,
9 file.h             <global>                                    20 struct path;
a file.h             <global>                                    21 extern struct file *alloc_file(struct path *, fmode_t mode,
b fs.h               <global>                                   977 struct path f_path;
c fs.h               <global>                                  1899 extern struct dentry *mount_subtree(struct vfsmount *mnt, const char
                                                                    *path);
d fs.h               <global>                                  1945 extern struct vfsmount *collect_mounts(struct path *);
e fs.h               <global>                                  1949 extern int vfs_statfs(struct path *, struct kstatfs *);

* 7814 more lines - press the space bar to display more *

Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
```

<Step 2> decach.h 파일에 들어가 봤더니 여러 개의 경로 관련 함수들이 존재했다.

```
/*
 * helper function for dentry_operations.d_dname() members
 */
extern char *dynamic_dname(struct dentry *, char *, int, const char *, ...);

extern char *__d_path(const struct path *, const struct path *, char *, int);
extern char *d_absolute_path(const struct path *, char *, int);
extern char *d_path(const struct path *, char *, int);
extern char *d_path_with_unreachable(const struct path *, char *, int);
extern char *dentry_path_raw(struct dentry *, char *, int);
extern char *dentry_path(struct dentry *, char *, int);
```

<Step 3> Find this C symbol 부분에 __d_path를 입력했다.

```
C symbol: __d_path

  File       Function         Line
0 dcache.h   <global>          343 extern char *__d_path(const struct path *, const struct path *, char *, int );
1 dcache.c   __d_path         2511 char *__d_path(const struct path *path,
2 seq_file.c seq_path_root     463 p = __d_path(path, root, buf, size);
3 path.c     d_namespace_path   86 res = __d_path(path, &root, buf, buflen);

Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
```

<Step 4> seq_file.c에 들어가서 CTRL+]을 사용해 __d_path함수 부분으로 갔다. 하지만 __d_path함수는 path의 root도 존재해야 하므로 다른 함수를 보기 위해 밑으로 가봤다. d_path함수가 존재했다. 이 함수 또한 path를 반환하는 함수로 찾고자 하는 것의 경로와 저장할 문자열, 문자열의 길이만 존재하면 절대경로를 찾아주기 때문에 적당해 보였다.

```
/**
 * __d_path - return the path of a dentry
 * @path: the dentry/vfsmount to report
 * @root: root vfsmnt/dentry
 * @buf: buffer to return value in
 * @buflen: buffer length
 *
 * Convert a dentry into an ASCII path name.
 *
 * Returns a pointer into the buffer or an error code if the
 * path was too long.
 *
 * "buflen" should be positive.
 *
 * If the path is not reachable from the supplied root, return %NULL.
 */
char *__d_path(const struct path *path,
               const struct path *root,
               char *buf, int buflen)
{
        char *res = buf + buflen;
        int error;

        prepend(&res, &buflen, "\0", 1);
        write_seqlock(&rename_lock);
        error = prepend_path(path, root, &res, &buflen);
        write_sequnlock(&rename_lock);

        if (error < 0)
                return ERR_PTR(error);
        if (error > 0)
                return NULL;
        return res;
}
"fs/dcache.c" [readonly] 3054L, 78758C
```

```
/**
 * d_path - return the path of a dentry
 * @path: path to report
 * @buf: buffer to return value in
 * @buflen: buffer length
 *
 * Convert a dentry into an ASCII path name. If the entry has been deleted
 * the string " (deleted)" is appended. Note that this is ambiguous.
 *
 * Returns a pointer into the buffer or an error code if the path was
 * too long. Note: Callers should use the returned pointer, not the passed
 * in buffer, to use the name! The implementation often starts at an offset
 * into the buffer, and may leave 0 bytes at the start.
 *
 * "buflen" should be positive.
 */
char *d_path(const struct path *path, char *buf, int buflen)
{
        char *res = buf + buflen;
        struct path root;
        int error;

        /*
         * We have various synthetic filesystems that never get mounted.  On
         * these filesystems dentries are never used for lookup purposes, and
         * thus don't need to be hashed.  They also don't need a name until a
         * user wants to identify the object in /proc/pid/fd/.  The little hack
         * below allows us to generate a name for these objects on demand:
         */
        if (path->dentry->d_op && path->dentry->d_op->d_dname)
                return path->dentry->d_op->d_dname(path->dentry, buf, buflen);
```

위의 단계를 거쳐 d_path함수를 발견한 뒤 이를 이용해 과제를 구현했더니 다음은 같은 결과가 나왔다.

```
[  154.096592] type=1400 audit(1480003035.590:43): apparmor="STATUS" operation="
profile_replace" name="/usr/lib/telepathy/telepathy-*" pid=2743 comm="apparmor_p
arser"
[  155.372499] init: plymouth-stop pre-start process (2843) terminated with stat
us 1
[  157.028178] init: vmware-tools pre-start process (2765) terminated with statu
s 1
[  389.090816]  ######### Loaded files of a process 'a.out(3612)' in VM ########
#
[  389.090821] (0x8048000~0x8049000) /home/os2014722057/Desktop/3-1/a.out
[  389.090824] (0x8049000~0x804a000) /home/os2014722057/Desktop/3-1/a.out
[  389.090827] (0x804a000~0x804b000) /home/os2014722057/Desktop/3-1/a.out
[  389.090829] (0xb75b1000~0xb7755000) /lib/i386-linux-gnu/libc-2.15.so
[  389.090832] (0xb7755000~0xb7757000) /lib/i386-linux-gnu/libc-2.15.so
[  389.090834] (0xb7757000~0xb7758000) /lib/i386-linux-gnu/libc-2.15.so
[  389.090837] (0xb776e000~0xb778e000) /lib/i386-linux-gnu/ld-2.15.so
[  389.090839] (0xb778e000~0xb778f000) /lib/i386-linux-gnu/ld-2.15.so
[  389.090841] (0xb778f000~0xb7790000) /lib/i386-linux-gnu/ld-2.15.so
[  389.090843] ################################$$################################
os2014722057@ubuntu:~/Desktop/3-1$
```

◆ 3-2

A. Introduction

3-2과제는 Bounded-buffer problem을 구현하는 것이다. Multi-thread 방식과 POSIX thread를 이용해서 구현한다. 또한 double linked list형태로 buffer를 구현한다. producer함수, consumer함수, insert함수, delete함수가 반드시 존재해야 하며 생산자, 소비자 thread가 종료될 때까지 계속해서 생산, 소비한다. 이때 발생할 수 있는 모든 동기화 문제를 해결해야 하며 생산자 수, 소비자 수, 최대 buffer 크기를 인자로 주어야 한다.

B. Conclusion

이번 과제를 구현하기 위해서 강의자료를 많이 참고해서 했다. main함수의 경우 실습자료에 있는 것을 토대로 구현했는데 이때 kernel창에 입력 받은 생산자 수, 소비자 수, 최대 buffer 크기를 atoi함수를 써 정수 형태로 바꿔 사용했다. 그리고 생산자와 소비자의 thread를 각각 만들기 위해 POSIX thread를 2개 만들어 사용했다. 이때 thread는 배열형태이다. 이를 통해 생산자 수에 맞게 생산자 thread를, 소비자 수에 맞게 소비자 thread를 만들었다. 각각 producer함수, consumer함수로 가 동작을 실행한다. producer함수와 consumer함수의 형태는 실습자료와 거의 비슷하지만 critical section이 다르다. critical section에 thread가 동시접속을 못하도록 시작부분에 mutex lock과 끝나는 부분에 mutex unlock을 했다. 우선 producer함수에서는 check_full함수를 통해 buffer가 full인지 아닌지 판단한 후 비어있는 공간이 존재하면 생산자 수를 나타내는 P_COUNT변수를 증가시킨다. 또한 insert함수를 써 새로운 buffer를 double linked list에 삽입한다. consumer함수에서는 check_empty함수를 통해 buffer가 emtpy인지 아닌지 판단한 후 채워진 공간이 존재하면 소비자 수를 나타내는 C_COUNT변수를 증가시킨다. 또한 delete함수를 써 root를 삭제해준다. count_producer함수도 따로 만들었다. 여기서는 buffer에 삽입된 생산자 중 마지막 생산자의 data_id를 반환하는 함수이다. 이를 통해 얼마나 생산했는지 확인할 수 있다. 생산자와 소비자는 소비 수와 생산 수를 나타내는 P_COUNT변수와 C_COUNT변수, buffer 정보를 공유한다. 이렇게 구현한 결과 다음과 같은 결과가 나왔다. 이때 계속해서 생산과 소비가 수행되므로 [Ctrl+c]를 눌러 중지시켰다.

```
os2014722057@ubuntu:~/Desktop/3-2$ make
gcc -o bbp bbp.c -lpthread -DMUTEX
os2014722057@ubuntu:~/Desktop/3-2$ ./bbp 3 4 5
[Producer 8598] produced 1 (buffer state : 1/5)
[Consumer 8602] consumed 1, produced from 8598 (buffer state : 0/5)
[Producer 8597] produced 2 (buffer state : 1/5)
[Producer 8596] produced 3 (buffer state : 2/5)
[Consumer 8601] consumed 2, produced from 8597 (buffer state : 1/5)
[Consumer 8599] consumed 3, produced from 8596 (buffer state : 0/5)
[Producer 8598] produced 4 (buffer state : 1/5)
[Consumer 8600] consumed 4, produced from 8598 (buffer state : 0/5)
[Producer 8597] produced 5 (buffer state : 1/5)
[Producer 8596] produced 6 (buffer state : 2/5)
[Consumer 8599] consumed 5, produced from 8597 (buffer state : 1/5)
[Consumer 8600] consumed 6, produced from 8596 (buffer state : 0/5)
[Producer 8598] produced 7 (buffer state : 1/5)
[Consumer 8602] consumed 7, produced from 8598 (buffer state : 0/5)
[Producer 8597] produced 8 (buffer state : 1/5)
[Producer 8596] produced 9 (buffer state : 2/5)
[Consumer 8599] consumed 8, produced from 8597 (buffer state : 1/5)
[Consumer 8602] consumed 9, produced from 8596 (buffer state : 0/5)
[Producer 8598] produced 10 (buffer state : 1/5)
[Consumer 8600] consumed 10, produced from 8598 (buffer state : 0/5)
^C
os2014722057@ubuntu:~/Desktop/3-2$
```