Operating System 실습
[Proc FileSystem]

한양대학교
HANYANG UNIVERSITY

Data-Centric Systems Lab.

Proc File System

- 프로세스 공간에 대한 접근을 목적으로 한 file system
- 리눅스의 proc file system은 커널 내의 정보를 표시하거나 커널 내의 파라미터 값을 표시하고 변경하는 등 다양한 목적을 위해 사용
- /proc/<process id>

함수명	내용
proc_mkdir	디렉토리 작성
proc_symlink	심볼 링크 작성
create_proc_entry	디렉토리 엔트리 작성(일반 파일 작성)

Proc File system

- /proc file system 내용의 일부
  - /proc/숫자 디렉토리
    * 현재 시스템에서 동작하는 프로세스의 정보가 포함
  - /proc/cpuinfo
    * CPU정보를 출력
  - /proc/filesystems
    * 시스템에서 지원하는 파일 시스템 목록을 보여준다(모듈포함)
  - /proc/kallsyms
    * EXPORT_SYMBOL
  - /proc/meminfo
    * 시스템 전체에 대한 메모리 사용 현황
  - /proc/mounts
    * 현재 마운트된 내용

Proc File system

- struct proc_dir_entry
  - void *data
    * read_proc이나 write_proc 필드에 선언된 함수에 전달할 데이터가 필요할 경우 이 필드를 이용해 데이터의 주소를 지정하여 전달할 수 있지만, 보통은 사용되지 않고 NULL로 지정
  - read_proc_t *read_proc
    * 생성된 proc 파일을 응용 프로그램에서 읽을 때 디바이스 드라이버가 파일 데이터를 제공하는 함수 주소를 지정
  - write_proc_t *write_proc
    * 응용 프로그램에서 proc 파일에 데이터를 써넣을 때 디바이스 드라이버가 이 데이터를 처리하기 위한 함수 주소를 지정

Proc File system

- int read_func(char* page, char** start, off_t off, int count, int* eof, void* data);

The read function should write its information into the page. For proper use, the function should start writing at an offset of off in page and write at most count bytes, but because most read functions are quite simple and only return a small amount of information, these two parameters are usually ignored (it breaks pagers like more and less, but cat still works).
If the off and count parameters are properly used, eof should be used to signal that the end of the file has been reached by writing 1 to the memory location eof points to. The parameter start doesn't seem to be used nywhere in the kernel. The data parameter can be used to create a single call back function for several files

- int write_func(struct file* file, const char* buffer, unsigned long count, void* data);

The write function should read count bytes at maximum from the buffer. Note that the buffer doesn't live in the kernel's memory space, so it should first be copied to kernel space with copy_from_user. The file parameter is usually ignored.

실습(myproc.c)

```c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>

#define EOF (0)


struct proc_dir_entry *dir_fp = NULL;
struct proc_dir_entry *B_fp = NULL;
struct proc_dir_entry *C_fp = NULL;

char strB[255];
char strC[255];


int copy_read_file(char *page, char **start, off_t off, int count, int *eof, void *data)
{
    int len;
    data = strB;

    len = sprintf(page, "this is copy \n %s", (char *)data);

    return len;
}


ssize_t write_file(struct file *filp, const char *buffer, unsigned long count, void *data)
{
    char *kdata;
    kdata = (char *)data;
    copy_from_user(kdata, buffer, count);
    kdata[count] = EOF;

    return count;
}
```

실습(cont'd)

```c
int read_file(char *page, char **start, off_t off, int count, int *eof, void *data)
{
        int len;
        len = sprintf(page, "%s\n", (char*)data);
        return len;
}

int module_start(void)
{
        dir_fp = proc_mkdir("myproc",0);
        B_fp = create_proc_entry("B", 0666, dir_fp);
        if(B_fp)
        {
                B_fp->data = strB;
                B_fp->read_proc = read_file;
                B_fp->write_proc = write_file;
        }
        C_fp = create_proc_entry("C", 0444, dir_fp);

        if(C_fp)
        {
                C_fp->data = strC;
                C_fp->read_proc = copy_read_file;
        }

        return 0;
}


void module_end(void)
{
        remove_proc_entry("C", dir_fp);
        remove_proc_entry("B", dir_fp);
        remove_proc_entry("myproc", 0);
}

module_init(module_start);
module_exit(module_end);
MODULE_LICENSE("GPL");
```

실습(Makefile)

```
obj-m := myproc.o

KDIR :=/lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

default :
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
clean :
    rm -rf *.ko
    rm -rf *.mod.*
    rm -rf .*.cmd
    rm -rf *.o
```

실습(cont'd)

- $cd /proc
- $ls //디렉토리 생성 확인
- $cd proc/myproc
- $ls –il //B,C 파일 생성 확인
- $cat C
- $cat B
- $echo "학번" > B
- $cat C

Grub 설정

- $uname -r
- $sudo vi /etc/default/grub

```
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
#GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""

# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"
```

- $sudo update-grub
- $sudo init 6