Linux File System – The procfs : /proc

Suntae Hwang
Kookmin University

The problem

- Modern kernel is highly complex
- Linux kernel has device drivers built-in
- An enormous amount of status information
- Many run-time configurable parameters
- How do we allow controlled access to kernel data and parameters and provide a familiar interface that programmers can easily adopt?

The Solution

- Create pseudo-filesystem to represent status information and configuration parameters as files
- Provides a unified “API” for collecting status information and configuring drivers
- Control access through UNIX permissions
- No new libraries needed ? simple filesystem calls are all that is necessary
- Quick, easy access via command line
- Not version- or configuration-specific

About procfs

- The idea of a Process Filesystem
  - Used for reporting process information only
  - Seen in UNIXes such as Solaris
- /proc extends the concept
- A similar implementation available forvarious flavors of BSD, including FreeBSD
- /proc for Linux is the most actively developed

/proc

- A virtual file system that provides information generated by the kernel on-the-fly
  - The files do not exist in physical storage device
- Used to
  - Configure kernel parameters
  - Look at kernel structures
  - Observe statistics from device drivers and general system information
- Mounted under /proc

Features

- /proc file system doesn't exist on any particular media.
- The contents of the /proc file system can be read by anyone who has the requisite permissions.
- Certain parts of the /proc file system can be read only by the owner of the process and of course root. (and some not even by root!!)
- The contents of the /proc are used by many utilities which grab the data from the particular /proc directory and display it.
- eg : top, ps, lspci ,dmesg etc

Tweak kernel parameters

- /proc/sys : Making changes in this directory enables you to make real time changes to certain kernel parameters.

- eg : /proc/sys/net/ipv4/ip_forward
  - It has default value of "0" which can be seen using 'cat'.
  - This can be changed in real time by just changing the value stored in this file from "0" to "1", thus allowing IP forwarding

Files in /proc

- loadavg
  - 지난 1,5,15 분간의 평균 시스템 부하
- uptime
  - 부트 이래로 총 가동 시간(초) 과 프로세스에 의해 사용된 총 시간
- meminfo
  - 메모리와 스왑의 사용 및 프리 바이트
- kmsg
  - 커널에 의해 읽혀질 커널 메시지
- version
  - 커널 또는/그리고 배포본의 버전
- cpuinfo
  - 프로세서(CPU) 파라메터
- pci
  - 현재의 PCI 슬롯 사용 정보
- self/
  - 현재 /proc를 억세스하고 있는 프로세스 정보
- net/
  - 네트워크 계층 설명
- scsi/
  - 개개의 scsi 장치에 관한 정보를 갖는 파일들이 위치함.
- malloc
  - kmalloc 과 kfree 운영 정보 모니터링
- kcore
  - 커널 코어 덤프
- modules
  - 단일 적재 모듈에 관한 정보
- stat
  - 일반 리눅스 통계

Files in /proc (cont'd)

- devices
  - 커널에 등록된 시스템 장치에 관한 정보
- interrupts
  - 인터럽트 할당 정보
- filesystems
  - 현재의 파일시스템 구현
- ksyms
  - 커널에의해 익스포트된 심볼
- dma
  - 사용중인 DMA 채널
- ioports
  - 현재 사용중인 입출력 포트
- smp
  - SMP가 활성화 돼 있을 경우 각 CPU 정보
- cmdline
  - 부트시에 커널에 넘겨진 파라메터
- sys/
  - 중요한 커널과 네트워크 정보
- mtab
  - 현재 마운트된 파일 시스템
- md
  - 다중 디바이스 드라이버 정보 (활성화 되었을 경우)
- rc
  - 확장 리얼타임 클럭
- locks
  - 현재 락(locked)된 파일
- Numerical named directories
  - 숫자로 된 디렉터리는 그 PID의 실행 프로세서 정보이다.

/proc/kcore - only the root user

- This file represents the physical memory of the system and is stored in the core file format.

- Unlike most /proc files, kcore does display a size. This value is given in bytes and is equal to the size of physical memory (RAM) used plus 4KB.

- Its contents are designed to be examined by a debugger, such as gdb, the GNU Debugger.

The numerical named directories

- The various directories in /proc are the processes that were running at the instant a snapshot of the /proc file system was taken.

- The contents of all the directories are the same as these directories contain the various parameters and the status of the corresponding process.

- You have full access only to the processes that you have started.

Files in the numerical named directories of /proc

- **cmdline**
  - it contains the whole command line used to invoke the process. The contents of this file are the command line arguments with all the parameters (without formatting/spaces).
- **cwd**
  - symbolic link to the current working directory
- **environ**
  - contains all the process-specific environment variables
- **exe**
  - symbolic link of the executable
- **maps**
  - parts of the process' address space mapped to a file.
- **fd/**
  - this directory contains the list file descriptors as opened by the particular process.
- **root**
  - symbolic link pointing to the directory which is the root file system for the particular process.
- **status**
  - information about the process.

Other Subdirectories in /proc

/proc/self/

- **link to the currently running process**

/proc/bus/

- **contains information specific to the various buses available on the system**
  - eg : for ISA, PCI, and USB buses, current data on each is available in /proc/bus/<bus type directory>
    - Individual bus directories, signified with numbers, contains binary files that refer to the various devices available on that bus
- **devices file**
  - USB root hub on the motherboard:

Subdirectories (cont...)

- **/proc/driver/**
  - **specific drivers in use by kernel**
    - rtc : output from the driver for the Real Time Clock
- **/proc/fs/**
  - **specific filesystem, file handle, inode, dentry and quota information**
- **/proc/ide/**
  - **information about IDE devices**
    - Each IDE channel is represented as a separate directory, such as /proc/ide/ide0 and /proc/ide/ide1
  - ***drivers* file**
    - version number of the various drivers
  - **Device directories**
    - data like cache, capacity, driver, geometry, media, model, settings

Subdirectories (cont...)

- **/proc/irq/**
  - **used to set IRQ to CPU affinity**
    - smp_affinity : which CPUs handle that specific IRQ
- **/proc/net/**
  - **networking parameters and statistics**
    - arp — kernel's ARP table. Useful for connecting hardware address to an IP address on a system.
    - dev — Lists the network devices along with transmit and receive statistics.
    - route — Displays the kernel's routing table.
- **/proc/scsi/**
  - **like /proc/ide it gives info about scsi devices**

/proc/sys

- allows you to make configuration changes to a running kernel
- Changing a value within a /proc/sys file is done by the 'echo' command
- Any configuration changes made thus will disappear when the system is restarted

/proc/sys subdirectories

- **/proc/sys/dev :**
  - **provides parameters for particular devices on the system**
    - cdrom/info : many important CD-ROM parameters
- **/proc/sys/fs**
- **/proc/sys/net**
- **/proc/sys/vm :**
  - **facilitates the configuration of the Linux kernel's virtual memory (VM) subsystem**

/proc/sys subdirectories (cont....)

/proc/sys/kernel

- **Acct**
  - Controls the suspension of process accounting based on the percentage of free space available on the filesystem containing the log
- **ctrl-alt-del**
  - Controls whether [Ctrl]-[Alt]-[Delete] will gracefully restart the computer using init (value 0) or force an immediate reboot without syncing the dirty buffers to disk (value 1).
- **domainname**
  - Allows you to configure the system's domain name, such as domain.com.
- **hostname**
  - Allows you to configure the system's host name, such as host.domain.com.
- **threads-max**
  - Sets the maximum number of threads to be used by the kernel, with a default value of 4095.
  - The random directory data related to generating random numbers for the kernel.
- **panic**
  - Defines the number of seconds the kernel will postpone rebooting the system when a kernel panic is experienced. By default, the value is set to 0, which disables automatic rebooting after a panic.

Testing for /proc

- /proc의 기능 및 그 내용들은 버전마다 다름

- /proc 내용 예

```
[root]/proc$ cat interrupts
                CPU0
      0:     36935631      XT-PIC  timer
      1:     2             XT-PIC  keyboard
      2:     0             XT-PIC  cascade
      3:     1             XT-PIC  serial
     10:     56316         XT-PIC  eth0
     11:     10            XT-PIC  aha1542
     13:     0             XT-PIC  fpu
     14:     382232        XT-PIC  ide0
    NMI:     0
```

Get memory information

$ more /proc/meminfo

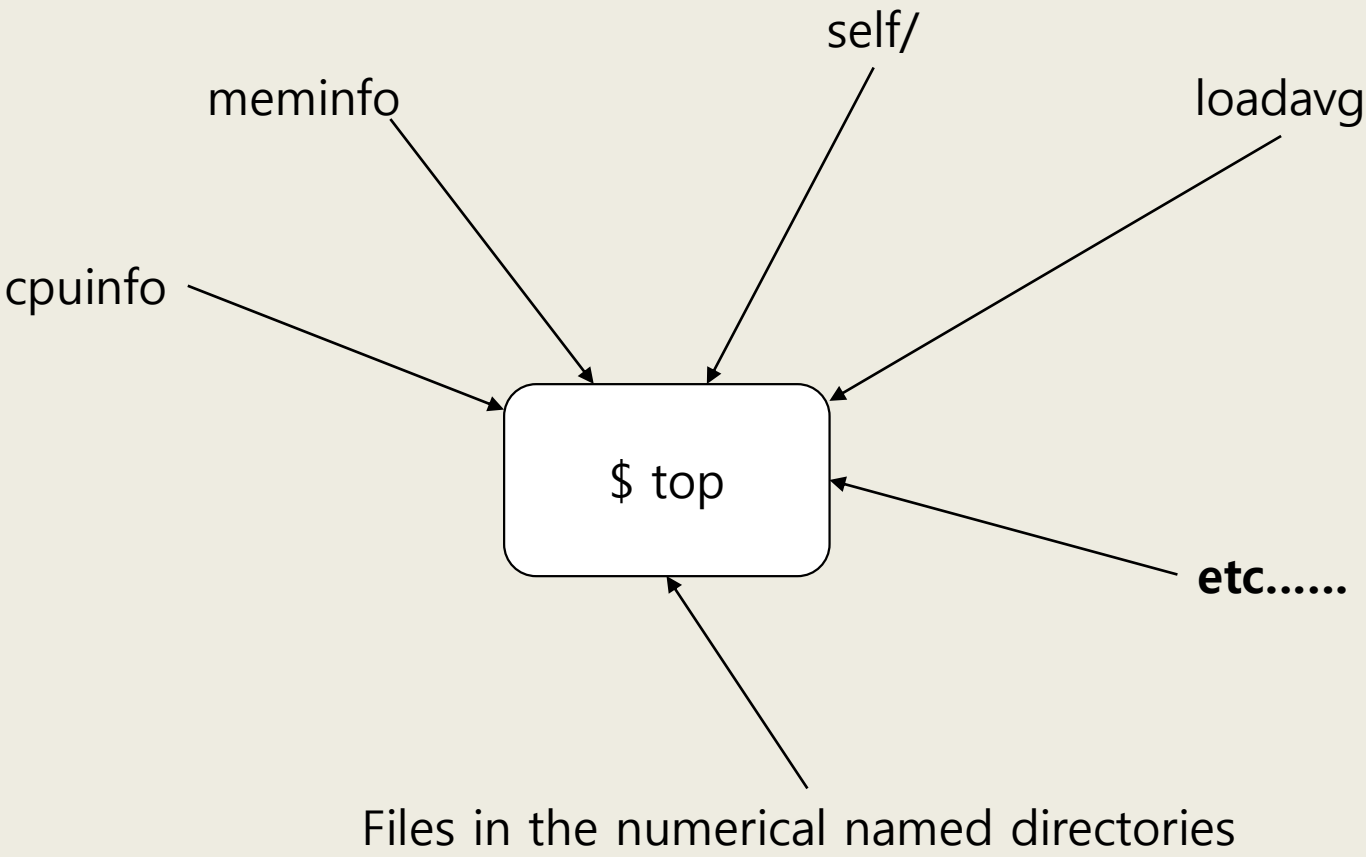
Get CPU information

$ more /proc/cpuinfo

Get kernel module information

```
$ more /proc/modules
```

Top command and /proc



Programming for /proc

- Simple filesystem representation allows for easy programming

- C calls
  - uptimefp = myfopen (PROC_DIR "uptime");
  - fgets (line, sizeof (line), uptimefp);
  - new.uptime = (unsigned long) (atof (strtok (line, " ")) * (unsigned long) HZ);

- Shell scripts ? bash, PERL, etc.

Advantages & Disadvantages

Advantages

- Coherent, intuitive interface to the kernel
- Great for tweaking and collecting status info
- Easy to use and program for

Disadvantages

- Certain amount of overhead, must use fs calls
- Alleviated somewhat by sysctl() interface
- User can possibly cause system instability

Next time

- **Presentation**
  - About result of homework
- **Reports**
- **Topics**
  - Kernel Module
  - How to handling procfs entries

Reports and Presentations

- **서식**
  - 1. Background
    - ps 명령을 구현하기 위해 필요한 /proc의 파일에 대한 학습 내용
  - 2. Main Idea
    - 구현하기 방안 제시 혹은 가설 제시
  - 3. Implementation
    - 구현을 통한 가설 검증 및 구현 방법에 대한 구체적 설명
  - 4. Conclusion
    - 과제에 대한 결론, 이 과제를 통해 얻은 점.

커널 모듈을 이용하여 procfs에 모듈 정보 표현

PROCFS AND KERNEL MODULE

procfs 프로그래밍

⊙ proc_dir_entry 구조체

```
struct proc_dir_entry {
    unsigned int low_ino;
    u8 namelen;
    char name[];
    mode_t mode;
    nlink_t nlink;
    uid_t uid; gid_t gid;
    unsigned long size;
    struct inode_operations * proc_iops;
    struct file_operations * proc_fops;
    get_info_t *get_info;
    struct module *owner;
    struct proc_dir_entry *next, *parent, *subdir;
    void *data;
    read_proc_t *read_proc;
    write_proc_t *write_proc;
    atomic_t count; /* use count */
    int deleted; /* delete flag */
    kdev_t rdev;
};
```

- **name:**
  - file name
- **data:**
  - pointer which can be used by proc handlers to pass local data
- **read_proc:**
  - read function pointer
- **write_proc:**
  - write function pointer

※ Kernel v3.18

procfs – kernel functions

Kernel Functions

- **proc_mkdir()**
  - Creates a directory ***name*** in the procfs directory ***parent***
- **proc_create()**
  - Creates a /proc file with the given name
  - To create a file in the root of the procfs, use **NULL** as ***parent*** parameter
- **create_proc_read_entry()**
  - create_proc_entry의 포장(wrapper)함수
- **create_proc_info_entry()**
  - create_proc_entry의 포장함수
- **proc_symlink()**
  - 심볼릭 링크를 만들기 위해서 사용
  - 실제 사용자(real user)만이 사용가능
- **remove_proc_entry()**
  - Removes the /proc file entry

procfs – kernel functions (2)

Kernel Functions (cont'd)

- **proc_net_create()**
  - create_proc_info_entry의 /proc/net정보에 대한 포장함수
  - 네트워크 서브시스템에 대해서 쉽게 접근하도록 도와줌

- **proc_net_remove()**
  - 네트워크 서브 시스템에 대한 remove_proc_entry의 포장함수

Exchange data between Kernel and User space.

⊙ 일반 유저와의 데이터 교환

- proc파일 시스템에서의 데이터는 실제 파일에 저장되는 것과는 달리 커널 메모리에 저장
- 유저가 데이터를 읽고 쓰기 위해서는 읽기와 쓰기를 위한 callback함수를 등록시켜서 사용 해야함
- proc_dir_entry에 읽기/쓰기를 위한 콜백함수를 등록

```
struct proc_dir_entry *entry;

struct file_operation operation;
operation.write = write_func;
operation.read = read_func;

entry = proc_create(NAME, PERMISSION, NULL, &operation);
```

Exchange data between Kernel and User space. (2)

- 데이터 읽기 (함수)

```
int read_func(struct file *file, char *user_buffer, size_t
count, loff_t *off);
```

- 데이터 쓰기 (함수)

```
int write_func(struct file* file, const char *user_buffer,
size_t count, loff_t *off);
```

Handling large data

⊙ Large Read Function

- **Supplied one page of memory as the first parameter to pass information to user space**
- **To read data larger than a page**
  - The kernel calls the proc read function multiple times
  - *offset* specifies the offset from where the read operation is requested
  - *count* specifies the number of byte to be read
  - *eof* is used to tell whether there is more data to be read