Operating Systems

Lab-3

‘/proc’ *filesystem in Linux*

/proc, in short for “process”, is a virtual filesystem that is created every time the system starts up. *It contains information related to the ongoing processes, memory management as well as some hardware configurations*.

Every Linux has a /proc filesystem no matter the type or version. Being a virtual filesystem, it can be accessed from any directory in Linux. To get inside the file-system, we run the command:

```
cd /proc
```

'cd' refers to a “change directory”, which is used to switch to other directories in Linux.

Contents of /proc file-system

Instead of changing directory, we can instead list all the files of /proc file-system on the terminal using:

```
ls /proc
```

‘ls’ command is used to list all files and directories present inside the specified location. The 'ls' command uses a color scheme for representation of files and directories.

- **Blue** – The blue part of the output represents **sub-directories**.
- White – The files that are uncolored are normal **files containing data**.
- **Cyan** – The cyan colored files are symbolic links.

As we can see /proc contains a huge number of files and directories. We will go through some important ones.

Numbered Directories

Each numbered directory denotes a Process ID. Process ID (PID) is a unique ID given a particular process that is either running or sleeping in the system. Each process directory contains files that store information about the respective process.

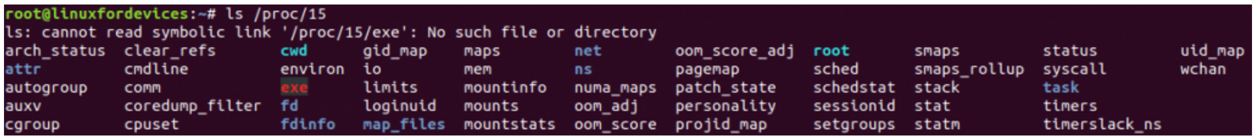
It must be noted that each process is crucial for the proper functioning of the system. Therefore, for complete access of each file in process directories, we need root access. It can be achieved by 'sudo -s' or 'sudo su' in Linux.

Let us look at an example for a process with PID = 15.

List of contents

Extracting the contents of the directory numbered 15, can be done by:

ls /proc/15



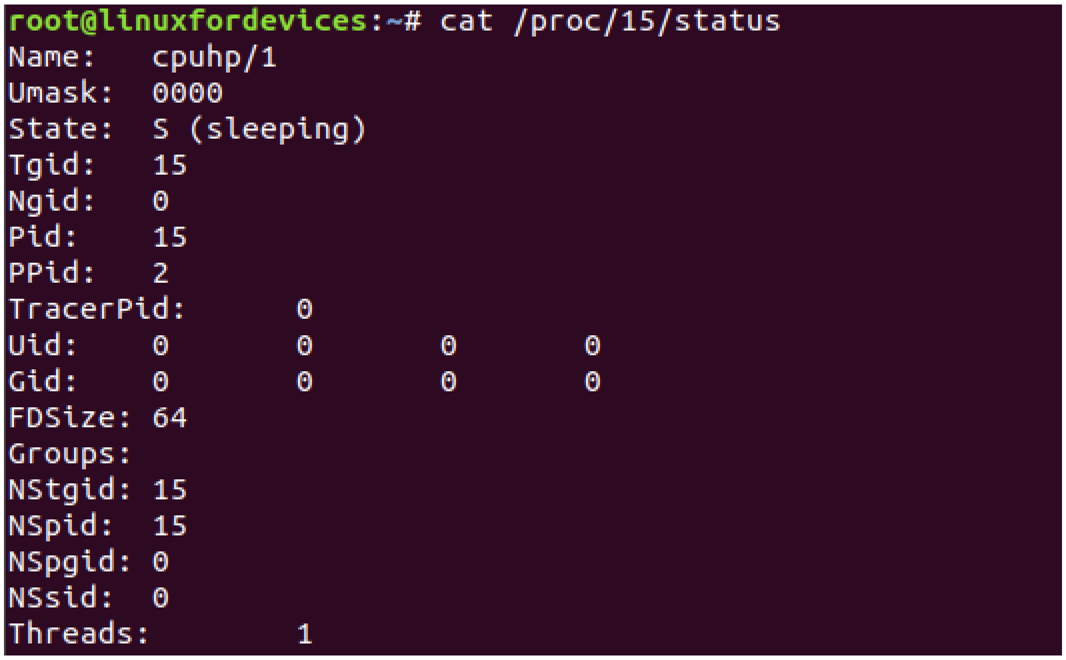
```
root@linuxfordevices:~# ls /proc/15
ls: cannot read symbolic link '/proc/15/exe': No such file or directory
arch_status  clear_refs       cwd      gid_map    maps        net        oom_score_adj  root       smaps         status         uid_map
attr         cmdline          environ  io         mem         ns         pagemap        sched      smaps_rollup  syscall        wchan
autogroup    comm             exe      limits     mountinfo   numa_maps  patch_state    schedstat  stack         task
auxv         coredump_filter  fd       loginuid   mounts      oom_adj    personality    sessionid  stat          timers
cgroup       cpuset           fdinfo   map_files  mountstats  oom_score  projid_map     setgroups  statm         timerslack_ns
```

Contents of directory ’15’

Process Information

To extract information regarding the process 15, we run:

cat /proc/15/status



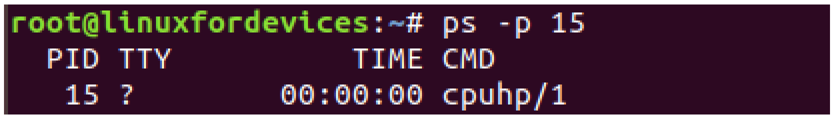
```
root@linuxfordevices:~# cat /proc/15/status
Name:   cpuhp/1
Umask:  0000
State:  S (sleeping)
Tgid:   15
Ngid:   0
Pid:    15
PPid:   2
TracerPid:      0
Uid:    0       0       0       0
Gid:    0       0       0       0
FDSize: 64
Groups:
NStgid: 15
NSpid:  15
NSpgid: 0
NSsid:  0
Threads:        1
```

Information about process ’15’

'cat' is a Linux tool for concatenating files. Here, we just used it to extract data stored in the 'status' file inside the '15' directory.

To verify the authenticity of the output, we can always check the process status using the ps command by:

ps -p 15


```
root@linuxfordevices:~# ps -p 15
  PID TTY          TIME CMD
   15 ?        00:00:00 cpuhp/1
```

Process status of ’15’ using ‘ps’

The above command filters out the process status according to the given PID.

Other details

Each file inside '/proc/15' contains some information related to process '15'. Some of the files are:

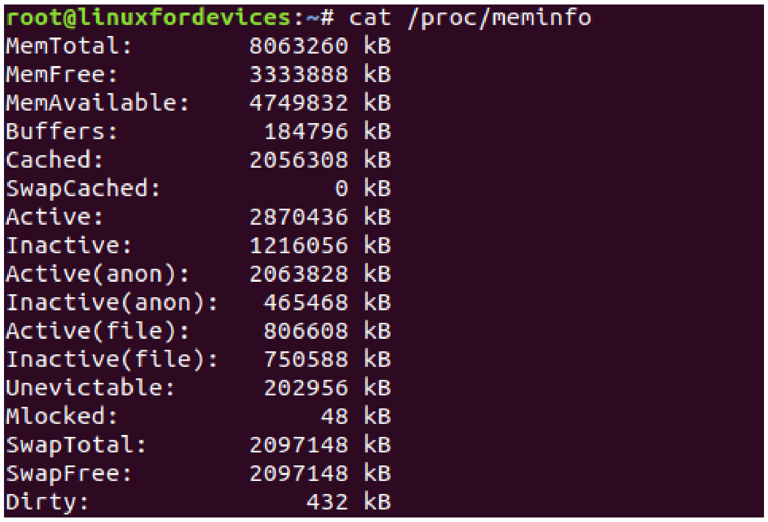
- **/proc/15/mem** – The **memory** the process already holds.
- **/proc/15/environ** – The **environment variables set** during the initiation of the process.
- **/proc/15/cwd** – The link to the **current working directory** (CWD) of the process.
- **/proc/15/limits** – Stores the values of **resource-limits** like CPU Time or Memory space.
- **/proc/15/fd** – The directory which contains **file descriptors**.
- **/proc/15/cmdline** – It contains the **whole command line** for the process.

To learn more about such files inside process-related directories, we can refer to the manual pages using 'man proc'.

Memory Statistics

‘/proc/meminfo’ contains information about the system’s memory usage. This file can be accessed by:

cat /proc/meminfo

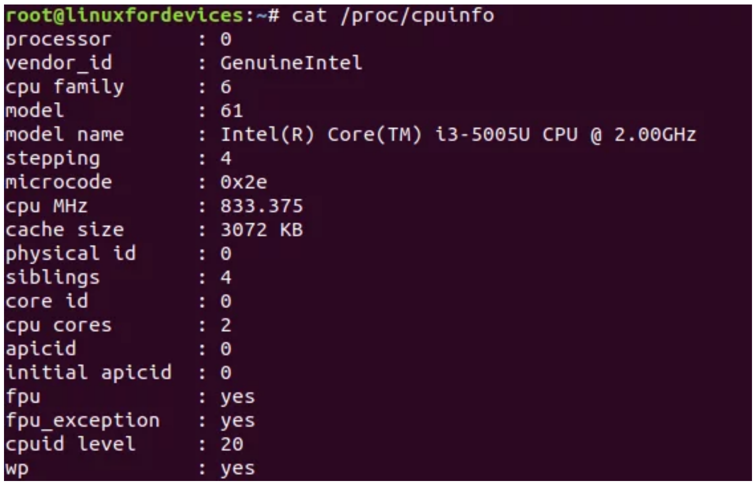

```
root@linuxfordevices:~# cat /proc/meminfo
MemTotal:        8063260 kB
MemFree:         3333888 kB
MemAvailable:    4749832 kB
Buffers:          184796 kB
Cached:          2056308 kB
SwapCached:            0 kB
Active:          2870436 kB
Inactive:        1216056 kB
Active(anon):    2063828 kB
Inactive(anon):   465468 kB
Active(file):     806608 kB
Inactive(file):   750588 kB
Unevictable:      202956 kB
Mlocked:              48 kB
SwapTotal:       2097148 kB
SwapFree:        2097148 kB
Dirty:               432 kB
```

Contents of ‘meminfo’ file

CPU Information

To access details related to CPU dependent items like CPU clock speed, model, etc, '/proc/cpuinfo' can be used:

cat /proc/cpuinfo



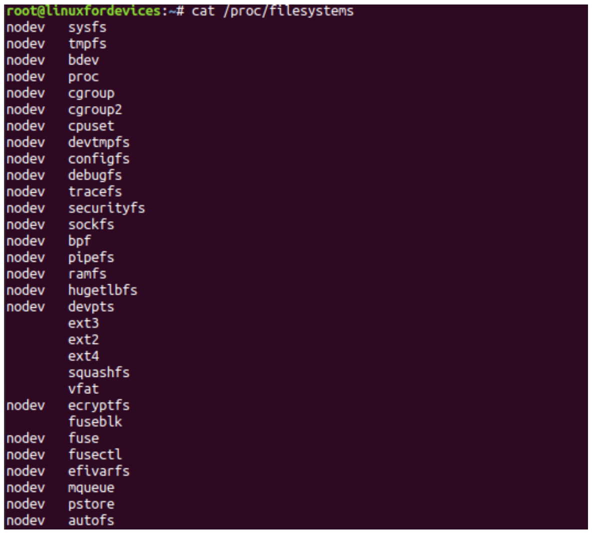
```
root@linuxfordevices:~# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 61
model name      : Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz
stepping        : 4
microcode       : 0x2e
cpu MHz         : 833.375
cache size      : 3072 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 20
wp              : yes
```

Contents of 'cpuinfo' file

Supported file-systems

'/proc/filesystems' contains the list of other file-systems currently supported or mounted by the Linux kernel.

cat /proc/filesystems



```
root@linuxfordevices:~# cat /proc/filesystems
nodev   sysfs
nodev   tmpfs
nodev   bdev
nodev   proc
nodev   cgroup
nodev   cgroup2
nodev   cpuset
nodev   devtmpfs
nodev   configfs
nodev   debugfs
nodev   tracefs
nodev   securityfs
nodev   sockfs
nodev   bpf
nodev   pipefs
nodev   ramfs
nodev   hugetlbfs
nodev   devpts
        ext3
        ext2
        ext4
        squashfs
        vfat
nodev   ecryptfs
        fuseblk
nodev   fuse
nodev   fusectl
nodev   efivarfs
nodev   mqueue
nodev   pstore
nodev   autofs
```

Other File-systems

The second column of the output contains the name of the file-systems supported, whereas the first column specifies whether it is currently mounted or not.

The use of 'nodev' means that the following file-system is not mounted.

Other files in ‘/proc’

Some of the other files containing important information are:

- /proc/interrupts – Contains interrupts for each CPU.
- /proc/ioports – Stores the list of all Input/Output ports in use.
- /proc/diskstats – Displays statistics for each disk device.
- /proc/version – Stores the kernel version.
- /proc/tty – Sub-directory containing files related to terminal drivers.

This article on proc file-system only touches the surface of the topic. It might be enough for a casual Linux user. In any case, if you still have a curiosity about the possibilities of the proc file-system, then you can make use of the man command (man proc).

Programming assignment #1

Building a /proc parser

The objective is to observe the OS through the /proc filesystem.

The OS is a program that uses various data structures. Like all programs in execution, you can determine the performance and other behavior of the OS by inspecting its state - the values stored in its data structures. In this part of the assignment, we study some aspects of the organization and behavior of a Linux system by observing values of kernel data structures exposed through the /proc virtual filesystem.

Linux uses the /proc file system to collect information from kernel data structures. The /proc implementation provided with Linux can read many different kernel data structures. If you cd to /proc on a Linux machine, you will see a number of files and directories at that location. Files in this directory subtree each correspond to some kernel data structure. The subdirectories with numeric names contain virtual files with information about the process whose process ID is the same as the directory name. Files in /proc can be read like ordinary ASCII files. You can open each file and read it using library routines such as fgets() or fscanf(). The proc manual page explains the virtual files and their content available through the /proc file system.

Requirements in detail:
In this part, you are asked to write a program to report the behavior of the Linux kernel. Your program should run in two different versions. The default version should print the following values on stdout:

- Processor type
- Kernel version
- The amount of memory configured into this computer
- Amount of time since the system was last booted

A second version of the program **should run continuously** and print lists of the following dynamic values (each value in the lists is the average over a specified interval):

- The percentage of time the processor(s) spend in user mode, system mode, and the percentage of time the processor(s) are idle
- The amount and percentage of available (or free) memory
- The rate (number of sectors per second) of disk read/write in the system
- The rate (number per second) of context switches in the kernel
- The rate (number per second) of process creations in the system

If your program (compiled executable) is called proc_parse, running it without any parameter should print out information required for the first version. Running it with two parameters "proc_parse <read_rate> <printout_rate>" should print out information required for the second version. read_rate represents the time interval between two consecutive reads on the /proc file system. printout_rate indicates the time interval over which the average values should be calculated. Both read_rate and printout_rate are in seconds. For instance, proc_parse 2 60 should read kernel data structures once every two seconds. It should then print out averaged kernel statistics once a minute (average of 30 samples). The second version of your program doesn't need to terminate.

Unix Documentation: The Manual
All Unix systems come with an online manual. The ”man" command is used to look up pages within the manual. For example, to look at the manual page for the ”chdir" system call, type:
man chdir

To look up information on the /proc virtual file system, type:
man proc

Sometimes, the same item appears in separate sections of the manual pages. System calls are documented in Section 2. You can specify a section number before the name of the item for which you want documentation. For example, to lookup the chdir() system call in Section 2, use the following command:
man 2 chdir

Section 1 contains information on commands, Section 2 contains information on system calls, and Section 3 contains information on C and Unix library functions.

For building this parser, you can use

- C/C++/Python programming skills, or
- even shell commands like grep, sed, awk, cut, etc, or

This is an exploratory project where you are free to employ any of the aforementioned methods to build the parser. Take note that this is an individual assignment. However, you are **encouraged** to help (and seek help from) your peers (except sharing code, of course).

A note on the programming language:
C/C++ is the preferred choice for this assignment. Most existing operating system kernels (Linux and other UNIX variants) themselves are written in C; the remaining parts are written in assembly language. Higher-level languages (Python, ...), while possible, are less desirable because C allows more flexible and direct control of system resources.

Turn-in:
You are asked to electronically turn in your source files (or a shell script). Attach a README file describing the names of your executables, the compiling instructions, or anything else special you want to let us know. The README file should be in **plain text** format. Instructions for electronic turn-in will be conveyed to you a few days before the deadline.

Deadline:
The deadline to complete this assignment is Sep 19, Sunday midnight.

Late turn-in policy:
Late turn-ins will be accepted for up to three days, with a 20% penalty for each late day. **No turn-ins more than three days late will be accepted.**

References:

- https://www.journaldev.com/41537/proc-file-system-in-linux
- https://www.cs.rochester.edu/users/faculty/sandhya/csc256/