1 SYSTEMS PROGRAMMING LABORATORY X - The /proc File System & Secure Programming

Examples&Exercises:

- Complete the following codes if necessary, then compile and run the code.
- Analyze the code and output.

1. CPU's clock speed; clock-speed.c
   - Study and analyze the code.
2. **/proc/self**; get-pid.c
   - Study and analyze the code.
   - Try to see the contents of the **/proc/self** entries by the commands **ls, cat**, etc.
3. Process Argument List; print-arg-list.c
   - Study and analyze the code.
   - Find a process having several arguments by giving the command **ps aux**, then use the process id of this process as the PID.

     ```
     $ ./print-arg-list PID
     ```
4. Write a program that
   - creates a zombie process,
   - shows the **cmdline** process directory entry.
5. Process Environment; print-environment.c
   - Study and analyze the code.
6. Process Executable get-exe-path.c
   - Study and analyze the code.
7. File descriptors; open-and-spin.c
   - Study and analyze the code.

```
$ ./open-and-spin open-and-spin.c (in one window)
$ ls -l /proc/PID/fd (in other window)
```

- In the first window, and find the process ID of the shell process by running **ps**.

  ```
  $ ps
  ```

- In the second window, and look at the contents of the *fd* subdirectory for that process.

  ```
  $ ls -l /proc/PID/fd
  ```

- In the second window, try writing a message to that file:

  ```
  $echo "Hello, world." >> /proc/PID/fd/1
  ```

8. System Statistics; print-uptime.c

   - Study and analyze the code.

9. Study the following commands;

```
$ ls -l /proc/version
$ cat /proc/version
$ man 5 proc
$ cat /proc/cpuinfo
$ strings -f /proc/[0-9]*/cmdline
$ cat /proc/version
$ cat /proc/sys/kernel/ostype
$ cat /proc/sys/kernel/osrelease
$ cat /proc/sys/kernel/version
$ cat /proc/meminfo
$ cat /proc/ide/ide?/hd?/media
$ cat /proc/ide/ide?/hd?/model
$ cat /proc/scsi/scsi
$ cat /proc/sys/dev/cdrom/info
$ cat /proc/uptime
$ cat /proc/devices
$ cat /proc/pci
$ cat /proc/dma
$ cat /proc/interrupts
$ cat /proc/ioports
$ cat /proc/net
$ cat /proc/kmsg
```

```
$ cat /proc/ksyms
$ cat /proc/modules
$ cat /proc/filesystems
$ cat /proc/partitions
$ cat /proc/mounts
$ cat /proc/loadavg
$ cat /proc/uptime
```

Which ones of these commands are for the process information, the hardware information and for the system statistics? These are not the all and try to see as much as possible entries in **/proc** file system.

10. TO BE GRADED; write a program that produces information by using the **/proc** filesystem for the followings:

    - the clock speed of the system's CPUs in MHz (not only the first CPU),
    - all the possible information about the process itself or a chosen process (with a known PID) in a human-readable format. *Hint:* The entries in the process directory (cmdline, cpu, cwd, environ, exe, fd, maps, mem, mounts, root, stat, statm, status).
    - the uptime and idle time for the system.

- Complete the following codes if necessary, then compile and run the code.
- Analyze the code and output.

1. Study the following commands;

```
$ id
$ man getuid
$ man getgid
$ man 2 stat (see the flags are defined for the st_mode field)
$ ls -ld /tmp
$ ls -la /tmp
$ man whoami
$ man su
$ ls -l /bin/su
```

2. Setuid Programs; setuid-test.c

   - Study and analyze the code.

   ```
   $ ./setuid-test
   $ chmod +s setuid-test
   $ ls -la
   $ ./setuid-test
   $ ls -la
   $ su
   $ ./setuid-test
   ```

3. Buffer Overruns;bufferoverflow.c

   - Study and analyze the code.

   ```
   ./bufferoverflow
   Please enter your name: 12345678whoami
   ```

4. Race Conditions in /tmp; temp-file.c

   - Complete the code,

5. Some Safer I/O Functions; symlink1.c Program illustrates a safer replacement for the chroot() and open() system calls and the fopen() standard library function. This illustrates the usual workaround, and wrappers for other system calls can be patterned after these functions.

   - Study and analyze the code.