Input/Output – 2

Week 12 – Lab

Virtual File Systems

- If you use **mount** command and look at your mount table, you will probably see that you have some filesystems mounted:
  - proc on */proc* type proc (rw,noexec,nosuid,nodev)
  - udev on */dev* type devtmpfs (rw,mode=0755)
  - */dev/sda2* on /boot type ext2 (rw)
  - */dev/sda1* on /boot/efi type vfat (rw)

Virtual Files

- Everything inside virtual filesystems acts like a file
- When a program (say, *cat*) wishes to open and read */proc/cpuinfo*, the filesystem driver for */proc/* answers the requests as if there were a file there, asks other parts of the kernel how many CPUs it is running on, and of what type, and answers the *read()* requests as if that were the contents of that file

Special Device Files

- It is possible to send data to a device or read data from a device by using ***cat*** command:
  - *cat /dev/audio > somefile.au*
    records an audio from microphone
  - *cat somefile.au > /dev/audio*
    plays recorded file

Input/Output (2)

- *cat /dev/random*
  Generates unlimited amount of random data
- */dev/full*
  Imitates disk full error (try **echo hi > /dev/full**)
- */dev/null*
  Discards all the data that are written. Returns EOF when a process tries to read from it
- */dev/zero*
  Returns as many null characters (0x00) as requested in *read()* call

Exercise 1

- Write a program that will generate and display a random string of 20 symbols using */dev/random* file. Save the code to *ex1.c* and the output to *ex1.txt*

tty

- "tty" actually stands for TeleType - the first type of device capable of acting as a terminal for a UNIX system. The name stuck, applying itself to the next step, right the way up to the present day
- In fact, the fancy graphical programs you use to talk to a shell prompt (xterm, konsole, PuTTY et al.) are called terminal emulators, because they copy ("emulate") a dumb terminal
- **echo hi > /dev/tty**

tee

- <command> | tee <list_of_sinks>
- Reads from standard input, and writes to standard output **and to files**
- Examples:
  - **date | tee file1 file2 file3**
    Date is written to three files and a screen
  - **echo hi | tee /dev/stdout**
    Prints “hi” twice

Disks

- */dev/sda* – primary master
- */dev/sdb* – primary slave
- */dev/sdc* – secondary master
- */dev/sdd* – secondary slave

Other Information

- *cat /proc/interrupts*
  Displays the number of interrupts per IRQ on the x86 architecture
- *cat /proc/iomem*
  Shows the current map of the system's memory for each physical device
- *cat /proc/ioports*
  Provides a list of currently registered port regions used for input or output communication with a device
- *cat /proc/version*
  Specifies the version of the Linux kernel, the version of gcc used to compile the kernel, and the time of kernel compilation

Exercise 2

- The **tee** command reads its standard input until end-of-file, writing a copy of the input to standard output and to the files named in its command-line arguments
- Implement **tee** using I/O system calls. By default, tee overwrites any existing file with the given name. Implement the **-a** command-line option (**tee -a file**), which causes tee to append text to the end of a file if it already exists
- Save the code to *ex2*.c and create *ex2.sh* that runs your program on */proc/cpuinfo* and saves the output to *ex2.txt*

Exercise 3 (Buffering)

- Using the **time** built-in command of the shell, try timing the operation of the program in *copy.c* (**time ./copy fileA fileB**)
  - Experiment with different file and buffer sizes. You can set the buffer size using the **-DBUF_SIZE=nbytes** option when compiling the program
  - Modify the *open()* system call to include the **O_SYNC** flag. How much difference does this make to the speed for various buffer sizes?
  - Save the results and explanation in file *ex3.txt*

References

- https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s2-proc-interrupts.html