Adding a new disk

The following commands assume that we have a new SATA disk attached to the virtual machine as described in the tutorial. Namely, the new disk is recognized as **sdb** in /dev. The following commands demonstrates what has to be done to make the disk usable.

```
$sudo mkfs –t ext4 /dev/sdb              # make file system of type “ext4”,
                                         # same as “format disk” in Windows

$sudo mkdir /NEW_DISK                    # create a mount point,
                                         # i.e. a folder that the new disk
                                         # will be accessed as a file tree,
                                         # not as a device

$sudo mount /dev/sdb /NEW_DISK           # finally, mount the disk.

$cd /NEW_DISK

$mount
… <skipped lines>…
/dev/sdb on /NEW_DISK type ext4 (rw,relatime,data=ordered)

$ls -alh
total 17K
drwxr-xr-x  3 root root 1.0K Nov  9 12:59 .
drwxr-xr-x 24 root root 4.0K Nov  9 17:34 ..
drwx------  2 root root  12K Nov  9 12:59 lost+found
```

Pay attention that root is the owner of the folder and that the permissions granted are 755.

Hard/Soft Links

In this session, we demonstrate how soft and hard links are used.
There is sample_text.txt file in the current directory. First, we make a hard link to this file.

```
$ls -alh
total 24K
drwxrwxr-x 2 eug eug 4.0K Nov  9 16:09 .
drwxrwxr-x 4 eug eug 4.0K Nov  9 16:08 ..
-rw-rw-r-- 1 eug eug    0 Nov  9 16:09 hs_link_demo.txt
-rwxrwxr-x 1 eug eug 7.2K Nov  9 15:24 lsof_demo
-rw-rw-r-- 1 eug eug  960 Nov  9 15:24 lsof_demo.c
-r--r--r-- 1 eug eug   38 Nov  9 09:20 sample_text.txt
```

```
$ln sample_text.txt sample_hl.txt
$ls -alh
total 28K
drwxrwxr-x 2 eug eug 4.0K Nov  9 16:09 .
drwxrwxr-x 4 eug eug 4.0K Nov  9 16:08 ..
-rw-rw-r-- 1 eug eug    0 Nov  9 16:09 hs_link_demo.txt
-rwxrwxr-x 1 eug eug 7.2K Nov  9 15:24 lsof_demo
-rw-rw-r-- 1 eug eug  960 Nov  9 15:24 lsof_demo.c
-r--r--r-- 2 eug eug   38 Nov  9 09:20 sample_hl.txt
-r--r--r-- 2 eug eug   38 Nov  9 09:20 sample_text.txt
```

Pay attention to the link counters of sample_text.txt and sample_hl.txt (equals 2 now).
Also both files are recognized as regular files (symbol '-' at the beginning) and have the same permissions 0444.
We change file permissions of one of the files. The expectation is that the permissions of the second one change too. The reason is that the permissions are kept in inode, which is the same for both files.

```
$ls -alh
total 28K
drwxrwxr-x 2 eug eug 4.0K Nov  9 16:09 .
drwxrwxr-x 4 eug eug 4.0K Nov  9 16:08 ..
-rw-rw-r-- 1 eug eug    0 Nov  9 16:09 hs_link_demo.txt
-rwxrwxr-x 1 eug eug 7.2K Nov  9 15:24 lsof_demo
-rw-rw-r-- 1 eug eug  960 Nov  9 15:24 lsof_demo.c
-r-------- 2 eug eug   38 Nov  9 09:20 sample_hl.txt
-r-------- 2 eug eug   38 Nov  9 09:20 sample_text.txt
```

Next, we create a soft link to the sample_text.txt file

```
$ln -s sample_text.txt sample_sf.txt
$ls -alh
total 32K
drwxrwxr-x 2 eug eug 4.0K Nov  9 16:09 .
drwxrwxr-x 4 eug eug 4.0K Nov  9 16:08 ..
-rw-rw-r-- 1 eug eug    0 Nov  9 16:09 hs_link_demo.txt
-rwxrwxr-x 1 eug eug 7.2K Nov  9 15:24 lsof_demo
-rw-rw-r-- 1 eug eug  960 Nov  9 15:24 lsof_demo.c
-r-------- 2 eug eug   38 Nov  9 09:20 sample_hl.txt
lrwxrwxrwx 1 eug eug   15 Nov  9 16:32 sample_sf.txt -> sample_text.txt
-r-------- 2 eug eug   38 Nov  9 09:20 sample_text.txt
```

We print out the content which is addressed by the soft link.

```
$cat sample_sf.txt
This is some text. Nothing special...
```

And we remove the original file

```
$rm sample_text.txt
rm: remove write-protected regular file 'sample_text.txt'? y
$ls -al
total 28
drwxrwxr-x 2 eug eug 4.0K Nov  9 16:09 .
drwxrwxr-x 4 eug eug 4.0K Nov  9 16:08 ..
-rw-rw-r-- 1 eug eug    0 Nov  9 16:09 hs_link_demo.txt
-rwxrwxr-x 1 eug eug 7.2K Nov  9 15:24 lsof_demo
-rw-rw-r-- 1 eug eug  960 Nov  9 15:24 lsof_demo.c
-r-------- 2 eug eug   38 Nov  9 09:20 sample_hl.txt
lrwxrwxrwx 1 eug eug   15 Nov  9 16:32 sample_sf.txt -> sample_text.txt
```

The soft link still “remembers” which file is addressed but there is no actual content available.

```
$cat sample_sf.txt
cat: sample_sf.txt: No such file or directory
```

Understanding /proc file system

In this demo we create a simple program in C, and check which files are opened.

First, the code.

```
$cat lsof_demo.c
/*----------------------------------------------------------------------
 * This program
 * 1. Opens a file for reading
 * 2. Sleeps for 60 seconds
 * 3. Closes the file and quits
 *
 ----------------------------------------------------------------------*/

#include <stdio.h>  //for printf
#include <unistd.h> //for sleep
#include <fcntl.h>  //for open
#include <errno.h>  //for errno
#include <string.h> //for strerror

int main(int argc, char** argv)
{
      int seconds_to_sleep = 60;
      const char* str_file_name = "./sample_text.txt";

      int sample_fd = open(str_file_name, O_RDONLY);
      if( 0 > sample_fd)
      {
            printf("Can't open %s\n", str_file_name);
            printf("Error - %s\n", strerror(errno) );
            return -1;
      }
      printf("%s is opened, file descriptor %d\n", str_file_name,
                                                  sample_fd);
      printf("Going to sleep for %d seconds\n", seconds_to_sleep);
      sleep(seconds_to_sleep);

      printf("Waking up\n");

      if( -1 == close(sample_fd) )
      {
            printf("Can't close %s\n", str_file_name);
            printf("Error - %s\n", strerror(errno) );
            return -1;
      }
      printf("Closing the file, quitting\n");
      return 0;
}
//=========================== END OF FILE ==============================
```

We compile it.

```
$gcc -o lsof_demo lsof_demo.c
```

We run it. Pay attention that the descriptor of the opened file is 3.
Pressing Ctrl+Z suspends the current running process.

```
$./lsof_demo
./sample_text.txt is opened, file descriptor 3
Going to sleep for 60 second
^Z
[1]+  Stopped                 ./lsof_demo
```

We put the current process to the background (the ***bg*** command) to determine its ID.

```
$bg
[1]+ ./lsof_demo &
```

The ***ps*** command prints the list of running processes.

```
$ps
  PID TTY          TIME CMD
 9009 pts/1    00:00:00 bash
 9061 pts/1    00:00:00 lsof_demo
 9070 pts/1    00:00:00 ps
```

We also check what is the name of the terminal we work in.

```
$tty
/dev/pts/1
```

What does /proc know about our process? (wide printouts are split into 2 lines)

```
$ls -al /proc/9061/fd
total 0
dr-x------ 2 eug eug  0 Nov  9 17:08 .
dr-xr-xr-x 9 eug eug  0 Nov  9 17:08 ..
lrwx------ 1 eug eug 64 Nov  9 17:08 0 -> /dev/pts/1
lrwx------ 1 eug eug 64 Nov  9 17:08 1 -> /dev/pts/1
lrwx------ 1 eug eug 64 Nov  9 17:08 2 -> /dev/pts/1
lr-x------ 1 eug eug 64 Nov  9 17:08 3 -> /home/eug/OS1617a …
                                             … /lab2/sample_text.txt
```

There are 3 soft links addressing the same device /dev/pts/1: 0 - stdin, 1 - stdout, 2 – stderr, which is the terminal we run in. Also, link #3 addresses the opened file. It is the same 3 as we saw previously is the program printout.

Now, we use lsof utility (LiSt Opened Files), which is another way to see either which files are opened by our process, or which process holds a particular file descriptor.
(wide printouts are split into 2 lines)

```
$lsof | grep lsof_demo
lsof_demo 9061  eug  cwd  DIR 8,1 4096    1050259 /home/eug/OS1617a/lab2
lsof_demo 9061  eug  rtd  DIR 8,1 4096          2 /
lsof_demo 9061  eug  txt  REG 8,1 7372    1048625 …
                                       … /home/eug/OS1617a/lab2/lsof_demo
lsof_demo 9061  eug  mem  REG 8,1 1802928 1181112 …
                                       … /lib/i386-linux-gnu/libc-2.24.so
lsof_demo 9061  eug  mem  REG 8,1 147656  1181086 …
                                       … /lib/i386-linux-gnu/ld-2.24.so
lsof_demo 9061  eug   0u  CHR 136,1  0t0        4 /dev/pts/1
lsof_demo 9061  eug   1u  CHR 136,1  0t0        4 /dev/pts/1
lsof_demo 9061  eug   2u  CHR 136,1  0t0        4 /dev/pts/1
lsof_demo 9061  eug   3r  REG   8,1   38  1049485 …
                                 … /home/eug/OS1617a/lab2/sample_text.txt
```

Some comments on the printouts above. File types in the 5th column: “REG” – regular file, “DIR” – directory, “CHR” – character device. File descriptors IDs are the integers in 4th column. A small letter near them: “r” – read only access, “u” – both read and write access.

```
$Waking up
Closing the file, quitting
[1]+  Done                    ./lsof_demo
```