CIS 452
Lab 13 Report

Ashley Hendrickson
Muna Gigowski
Fall 2019

File Locking

1. Carefully compare the modified source code file to the original Sample Program 1. What *exactly* did Sample Program 1 do?

The sample program begins by declaring a struct of type flock (to hold the data for fcntl later), as well as a char array of size 30, which holds the string "// is this Sample Program 1?". The program then checks to ensure a filename argument was provided, then attempts to open the given file with the O_RDWR flag (which indicates a read/write access mode). Upon successfully opening the file, the program then begins filling the flock struct with the appropriate properties to specify the type of lock and the start and end locations for the lock….Then the program attempts to call fcntl() on the file with the newly created flock struct and the F_SETLK command to aquire the file lock. If the lock call is successful, the program then writes the buf string to the file, sleeps for 10 seconds, closes the file, then returns.

In calling this program on itself, the result is that the comment at the top of the original source code file that read "// this is Sample Program 1!" got changed to the string that was declared in the char array, buf, earlier: "// is this Sample Program 1?"

2. What happened, and why?

When we started up the immediate second run of the program in the second terminal, the program output the error message "no way: resource temporarily unavailable." This happened because the first instance of the program had already placed a lock on the file, so the second instance of the program could not access that file at the same time. Thus, when the second instance tried to execute fcntl (fd, F_SETLK, &fileLock), it failed and output the error message.

3. Submit your modified programs (or at least the relevant lines of code in each).

<u>SAMPLEPROGRAM1A.C CODE:</u>

```
// this is Sample Program 1!

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>

#define SIZE 30

int main(int argc, char *argv[])
{
  struct flock fileLock;
```

```
   int fd;
   char buf[SIZE] = "// is this Sample Program 1?";

   if (argc < 2) {
      printf ("usage: filename\n");
      exit (1);
   }
   if ((fd = open (argv[1], O_RDWR)) < 0) {
      perror ("there is");
      exit (1);
   }

   fileLock.l_type = F_WRLCK;
   fileLock.l_whence = SEEK_SET;
   fileLock.l_start = 0;
   fileLock.l_len = 0;
   if (fcntl (fd, F_SETLK, &fileLock) < 0) {
      perror ("no way");
      exit (1);
   }

   write (fd, buf, SIZE-2);

   sleep (10);

   if (fcntl (fd, F_UNLCK, &fileLock) < 0) {
      perror ("error unlocking file");
      exit (1);
   }

   close(fd);

   return 0;
}
```

<u>SAMPLEPROGRAM1B.C CODE:</u>

```
// this is Sample Program 1!

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>

#define SIZE 30

int main(int argc, char *argv[])
```

```
{
   struct flock fileLock;
   int fd;
   char buf[SIZE];

   if (argc < 2) {
      printf ("usage: filename\n");
      exit (1);
   }
   if ((fd = open (argv[1], O_RDWR)) < 0) {
      perror ("there is");
      exit (1);
   }

   fileLock.l_type = F_WRLCK;
   fileLock.l_whence = SEEK_SET;
   fileLock.l_start = 0;
   fileLock.l_len = 0;
   if (fcntl (fd, F_SETLKW, &fileLock) < 0) {
      perror ("no way");
      exit (1);
   }

   read (fd, buf, SIZE);
   printf("%s", buf);

   sleep (10);

   close(fd);

   return 0;
}
```

Linking

4. Give two ways to tell that a file is actually a symbolic link to another file

One way to tell that a file is a symbolic link to another file is that the ls -l command displays hard links by their name alone, while soft links are displayed with arrows to the actual files that they point to. For example:

```
softjunk -> forlinking/junk.c
```

Another way is that, when looking at the file permissions shown by ls -l, the soft link's permissions are preceded by a lowercase "l," implying again that it is a soft link. Example:

```
lrwxrwxrwx
```

5. Why are the two link counts different?

The links counts are different because the "link count" value is the number of different directory entries that all point to the inode associated with the object. Since the original file itself is a directory entry that points to its own inode, the hard link we created becomes the second directory entry to point to that same inode - that is why the link count displayed beside hardjunk is 2. However, since symbolic links do not point to the same inode as the original file they reference, and instead point to a new inode, they are the only directory entry pointing there so the link count beside softjunk is 1.

6. What are the reported sizes of the 3 files? Why are the two link file sizes different?

Original file size: 739 bytes

Hard link size: 739 bytes

Symbolic link size: 17 bytes

The two link file sizes are different because the hard link points to the same inode as the original file, so their sizes will be the same because ls returns the size of the file itself, but for the symbolic link ls just returns the size of the path to the file the symbolic link is referencing - which is much smaller.

7. What happened when you tried to display the link files? Explain, please.

After we deleted the original file, when we went to display the contents of hardjunk they still displayed the same way as before because hardjunk is still pointing to the inode that the original file “junk.c” was also pointing to - even after junk.c got deleted the inode still exists. However when we tried to display the contents of softjunk the terminal returned an error stating “No such file or directory.” This is because softjunk was pointing to an inode that contained the file path of the original junk.c file, so when we deleted that file its file path no longer existed.

Mini-Programming Assignment (my_ln)

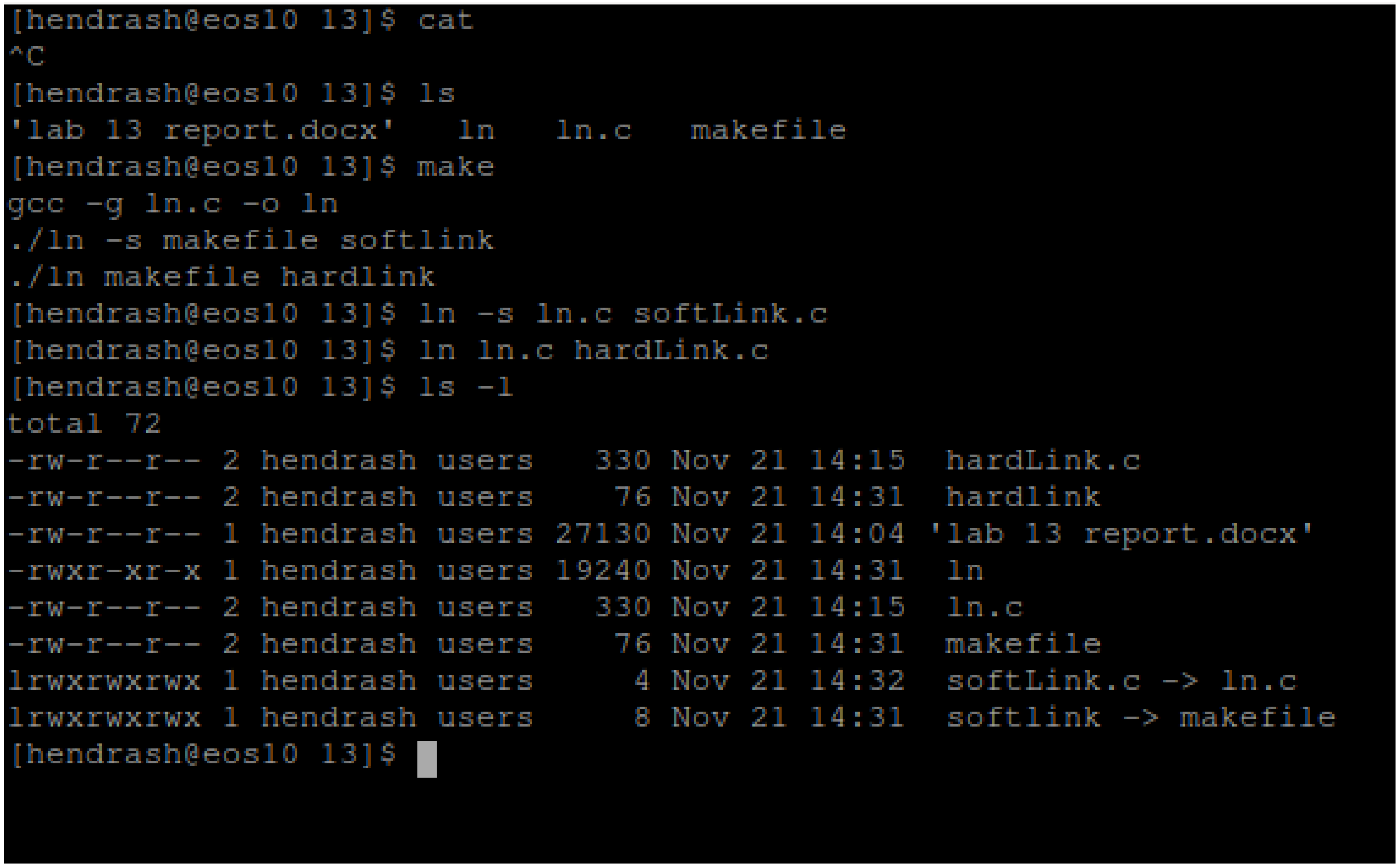
LINKPROGRAM.C SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <errno.h>
#include <dirent.h>
#include <string.h>
#include <unistd.h>
```

```
#include <unistd.h>

int main(int argc, char *argv[])
{           if(strcmp(argv[1], "-s")==0 ){
            symlink(argv[2],argv[3]);
                                                 }
else{
link(argv[1], argv[2]);   }
            return 0;
}
```

SAMPLE OUTPUT SCREENSHOT:



```
[hendrash@eos10 13]$ cat
^C
[hendrash@eos10 13]$ ls
'lab 13 report.docx'   ln   ln.c   makefile
[hendrash@eos10 13]$ make
gcc -g ln.c -o ln
./ln -s makefile softlink
./ln makefile hardlink
[hendrash@eos10 13]$ ln -s ln.c softLink.c
[hendrash@eos10 13]$ ln ln.c hardLink.c
[hendrash@eos10 13]$ ls -l
total 72
-rw-r--r-- 2 hendrash users   330 Nov 21 14:15  hardLink.c
-rw-r--r-- 2 hendrash users    76 Nov 21 14:31  hardlink
-rw-r--r-- 1 hendrash users 27130 Nov 21 14:04 'lab 13 report.docx'
-rwxr-xr-x 1 hendrash users 19240 Nov 21 14:31  ln
-rw-r--r-- 2 hendrash users   330 Nov 21 14:15  ln.c
-rw-r--r-- 2 hendrash users    76 Nov 21 14:31  makefile
lrwxrwxrwx 1 hendrash users     4 Nov 21 14:32  softLink.c -> ln.c
lrwxrwxrwx 1 hendrash users     8 Nov 21 14:31  softlink -> makefile
[hendrash@eos10 13]$
```