Lauren Fromm
404751250

Lab 4 Log

First, I need to import the coreutils package onto my system.
```
$ wget http://web.cs.ucla.edu/classes/fall17/cs35L/
       assign/coreutils-with-bug.tar.gz
```
This imports the package with the bug.

Next, I want to unzip the tarbel.
```
$ tar -xzvf coreutils-with-bug.tar.gz
```
This outputs all the files that are being unzipped.

I then switch into the directory:
```
$ cd coreutils-with-bug
```
Now, I want to configure and make the file.
I use --prefix to put it into a new folder.
```
$ ./configure --prefix='/u/cs/ugrad/jeanne/cs35l/lab4/coreutils'
```
This outputs a lot of checks. I then run make:
```
$ make
```
After running make, there is an error:
```
In file included from utimecmp.c:41:0:
utimens.h:2:5: error: conflicting types for 'futimens'
 int futimens (int, char const *, struct timespec const [2]);
     ^
In file included from utimecmp.h:25:0,
                 from utimecmp.c:25:
/usr/include/sys/stat.h:373:12: note: previous declaration of
'futimens' was here
 extern int futimens (int __fd, const struct timespec __times[2])
__THROW;
```

We are given a patch from the spec.
I download the patch by using:
```
$ wget http://web.cs.ucla.edu/classes/fall17/cs35L/assign/
coreutils.diff
```
Which downloads coreutils.diff into my directory.

I want to move coreutils.diff into coreutils-with-bug:
```
$ mv coreutils.diff ~/cs35l/lab4/coreutils-with-bug/coreutils.diff
```

I now want to apply the patch by using:
```
$ patch -p0 < coreutils.diff
```
This outputs the files that were patched:
```
patching file lib/utimens.c
patching file lib/utimens.h
patching file src/copy.c
patching file src/tee.c
patching file src/touch.c
```

I then run make again:
```
$ make
```
And now there are no errors.
The patched worked because the error occured because futimens was reused as a name, so there was a conflict of types. The patch renamed one of the futimens to coreutils_futimens so there was no confusion, and no error was thrown.

I know use make install to put the install into my new folder coreutils
```
$ make install
```
Now if I go back:
```
$ cd ..
```
I can see that there is a new folder called coreutils that has the patched file.

I now want to reproduce the problem.
First I go into the source folder:
```
$ cd ~/cs35l/lab4/coreutils-with-bug/src
```
Then I follow the steps given:
```
$ tmp=$(mktemp -d)
$ cd $tmp
$ touch -d '1918-11-11 11:00 GMT' wwi-armistice
$ touch now
$ sleep 1
$ touch now1
$ ~/cs35l/lab4/coreutils-with-bug/src/ls -lt --full-time wwi-armistice
now now1
```
This outputs:
```
-rw-r--r-- 1 jeanne csugrad 0 1918-11-11 03:00:00.000000000 -0800 wwi-
armistice
-rw-r--r-- 1 jeanne csugrad 0 2017-10-26 14:09:23.143762091 -0700 now1
-rw-r--r-- 1 jeanne csugrad 0 2017-10-26 14:09:18.704641871 -0700 now
```
This is the buggy version because we see that the order is opposite of what I want, I want the most recent file at the top.

I then want to enter the GDB
```
$ gdb ./ls
```
I then need to find what to use to effectivly debug.
```
(gdb) info functions
```
The functions that sound interesting are:
```
static int compare_atime(V, V);
static int compare_ctime(V, V);
static int compstr_mtime(V, V);
static void sort_files(void);
```
I start by placing a breakpoint at sort_files.
```
$ break sort_files
```
which outputs:
```
Breakpoint 1 at 0x4040d0: file ls.c, line 2954.
```

I then run the command by using:
```
(gdb) run -lt wii-armistice now now1
```
I step through until I find an interesting switch statement.
Since I know -t sorts files by mtime, I focus on the mtime line:
```
case time_mtime:
     func = sort_reverse ? rev_cmp_mtime : compare_mtime;
```
I want to look at compare_mtime to make sure that reversal is not happening
```
(gdb) break compare_mtime
```
When I get to compare_mtime, I use:
```
(gdb) list
```
which shows me that compare_mtime calls another function, cmp_mtime.
So, I set a new breakpoint
```
(gdb) break cmp_mtime
```
When compare_mtime is called I use:
```
(gdb) s
```
To step into the function and see the value that is returned.
I then use:
```
(gdb) list
```
To see the rest of the function. I see:
```
43 /* Return negative, zero, positive if A < B, A == B, A > B,
respectively.
44    Assume the nanosecond components are in range, or close to it.
*/
45    static inline int
46    timespec_cmp (struct timespec a, struct timespec b)
47    {
48      int diff = a.tv_sec - b.tv_sec;
49        return diff ? diff : a.tv_nsec - b.tv_nsec;
50        }
51
52        # if ! HAVE_DECL_NANOSLEEP
```

So then I use print diff to look at the value of diff:
```
(gdb) print diff
```
which outputs;
```
$6 = -1613826000
```
This value seems to be unreliable, so I think this is where the problem is.
The problem occurs when there is integer overflow when a.tv_sec is too small.
I see that this function is in timespec.h:
```
timespec_cmp (b=..., a=...) at ../lib/timespec.h:48
```
So, I change directories to find this file:
```
(gdb) quit
$ cd ~/cs35l/lab4/coreutils-with-bug/lib
```
I use ls to make sure the file is in this folder, which it is.
I then copy the file into my lab directory so that I can make changes:
```
$ cp timespec.h ~/cs35l/lab4/timespec.h
$ cd timespec.h ~/cs35l/lab4
```

```
Then I edit the file:
$ emacs timespec.h
To avoid integer overflow, we use just simple compare commands:
  if (a.tv_sec < b.tv_sec)
    return -1;
  else if (a.tv_sec > b.tv_sec)
    return 1;
  else
    return 0;
I then want to make my diff file, so I use:
$ diff -u ~/cs35l/lab4/coreutils-with-bug/lib/timespec.h ~/cs35l/lab4/
timespec.h
  > lab4.diff
Which puts it in lab4.diff for me.
To get my changelog entry, I reenter timespec.h
$ emacs timespec.h
$ C-x 4 a
I copy and past the changelog into my diff file.

To patch the file, I first make a copy of my diff file.
$ cp lab4.diff ~/cs35l/lab4/coreutils-with-bug/timepatch.diff
$ mv timepatch.diff lib
I then want to remove some of the lengthy file names in the patch so
that the patch
works corretly. I run the patch using:
$ patch -p1 < timepatch.diff
I remake the directory:
$ make
I then run the commands to remake a temporary directory like before:
$ tmp=$(mktemp -d)
$ cd $tmp
$ touch -d '1918-11-11 11:00 GMT' wwi-armistice
$ touch now
$ sleep 1
$ touch now1
$ ~/cs35l/lab4/coreutils-with-bug/src/ls -lt --full-time wwi-armistice
now now1
Which gives:
-rw-r--r-- 1 jeanne csugrad 0 2017-10-26 20:58:33.346403410 -0700 now1
-rw-r--r-- 1 jeanne csugrad 0 2017-10-26 20:58:29.806307473 -0700 now
-rw-r--r-- 1 jeanne csugrad 0 1918-11-11 03:00:00.000000000 -0800 wwi-
armistice
So the patch worked!

To test on the home directory:
$ touch -d '1918-11-11 11:00 GMT' wwi-armistice
$ touch now
$ touch now1
$ ls -lt --full-time wwi-armistice now now1
Which gives:
```

```
-rw-r--r-- 1 jeanne csugrad 0 2054-12-17 09:28:16.000000000 -0800 wwi-
armistice
-rw-r--r-- 1 jeanne csugrad 0 2017-10-26 21:00:16.972865000 -0700 now1
-rw-r--r-- 1 jeanne csugrad 0 2017-10-26 21:00:14.005596000 -0700 now
```

For some reason, the linux server changed 1918-11-11 to 2054-12-17.
After doing some research online, I find that this happens because
In linux, the Unix time which describes as the number of seconds
that have elapsed since January 1st, 1970. The linux sysytem uses
signed 32-bit time stamps so that dates before 1970 can be
represented.
The SEASnet system uses unsigned 32-bit time stamps, so the sign
bit is lost, and the date is shown in the future instead of
the actual date.

Even though the date is not correct, the dates are still sorted from
newest to oldest.