Assignment 3. Modifying and rewriting software

Laboratory: Installing a small change to a big package

Keep a log in the file `lab3.txt` of what you do in the lab so that you can reproduce the results later. This should not merely be a transcript of what you typed: it should be more like a true lab notebook, in which you briefly note down what you did and what happened.

You're helping to build an application containing a shell script that invokes the `ls` command to get file status. Your application is running atop the Maroon Chapeau Enterprise Linux 5 distribution, which uses the `ls` implementation supplied by Coreutils 7.6. You've been running into the problem that for some users `ls` generates output that looks like this:

```
$ ls -l /bin/bash
-rwxr-xr-x 1 root root 729040 2009-03-02 06:22 /bin/bash
```

The users want the traditional Unix format, which looks like this:

```
$ ls -l /bin/bash
-rwxr-xr-x 1 root root 729040 Mar  2  2009 /bin/bash
```

You've been asked to look into the problem and fix it.

You discover that the problem is that in some cases users set their locale to a value like `en_US.UTF-8`, for example, by setting the `LC_ALL` environment variable to that value:

```
$ export LC_ALL='en_US.UTF-8'
```

Users who have done this get the YYYY-MM-DD date instead of the traditional Unix date.

You nose around on the net, and discover that the problem is that the locale files for Coreutils are not generated properly (see Jim Meyering's message of 2009-09-29, also archived in case the primary web site is down). Getting these files generated and distributed to all your clients seems like a bit of a hassle, so instead, you decide to patch the `ls` program instead, using a temporary workaround patch published by Pádraig Brady (also archived).

Try Brady's workaround, as follows:

1. Grab Coreutils 7.6.
2. Compile and install your copy of Coreutils into a temporary directory of your own. Note any problems you run into.
3. Reproduce the bug on your machine with the unmodified version of coreutils. You may need to use the `locale-gen` program to generate the `en_US.UTF-8` locale.
4. Use Emacs or Vim to apply Brady's patch.
5. Type the command `make` at the top level of your source tree, so that you build (but do not install) the fixed version. For each command that gets executed, explain why it needed to be executed (or say that it wasn't neeeded).
6. Make sure your change fixes the bug, by testing that the modified `ls` works on your test case and that the installed `ls` doesn't. Test on a file that has been recently modified, and on a file that is at least a year old. You can use the `touch` command to artficially mark a file as being a year old.

Q1. Why did Brady's patch remove the line "`case_long_iso_time_style:`"? Was it necessary to remove that line? Explain.

Q2. If your company adopts this patched version of Coreutils instead of the default one, what else should you watch out for? Might this new version of Coreutils introduce other problems with your application, perhaps in countries where users don't speak English and don't understand English-format dates?

Homework: Rewriting a script

Consider the Python script `randline.py`.

Q3. What happens when this script is invoked on an empty file like `/dev/null`, and why?

Q4. What happens when this script is invoked with Python 3 rather than Python 2, and why? (You can run Python 3 on the SEASnet hosts by using the command `python3` instead of `python`.)

Write a new script `comm.py` in the style of `randline.py`; your script should implement the POSIX `comm` command.

Your implementation should support all the options and operands required by POSIX. For example, if the file `words.txt` and standard input are both text files sorted in the current locale, `comm -12 words.txt -` should output a sorted file containing all lines that are in both `words.txt` and standard input. Your implementation should also support all the environment variables required by POSIX, except that some or all of its diagnostics can use English regardless of the values of the`LC_MESSAGES` and `NLSPATH` environment variables. As with the original `randline.py`, your program should report an error if given the wrong number of input operands.

Your implementation should also support an extra option `-u`, which causes `comm` to work even if its inputs are not sorted. Output lines should appear in the same order as the input lines. If a line appears in both input files, it should be output according to the first input file's order. Lines that appear only in the second input file should be output after all other lines.

Your implementation of `comm.py` should use only the `locale` and `string` modules and the modules that `randline.py` already uses (it should not import any other modules). Don't forget to change its usage message to accurately describe the modified behavior.

Port your `comm.py` implementation to Python 3. Make sure that it still works with Python 2. Don't rewrite it from scratch; make as few changes as is reasonable.

Submit

Submit the following files.

- The file `lab3.txt` as described in the lab.
- A file `hw3.txt` containing the answer to questions Q1 through Q4 noted above.
- The file `comm.py` as described in the homework.

All files should be ASCII text files, with no carriage returns, and with no more than 80 columns per line. The shell command:

```
expand lab3.txt hw3.txt comm.py | awk '/\r/ || 80 < length'
```

should output nothing.