

Major Program 1: Cooperating processes and pipes

Due: Thursday, Feb 9 at 11:59 pm

Overview

This assignment is designed to help you become proficient in the use of fork, wait, and pipes. Your mission is to build a filter that will extract certain e-mail addresses from within a large file of e-mails. An example large e-mail file is to be found in the class directory and is named fudan.txt

Your program is to read an e-mail file <<<from the standard input>>> and create three output files:

- 1 - discard.1 - Contains all lines in the input file that <<don't>> contain the word **From** (letter case IS significant).
- 2 - discard.2 - Contains all of the email addresses in the file that contain the strings Clemson, clemson, or CLEMSON
- 3 - std output - Contains all of other email addresses in the file.

Your main process should create 2 pipes and the fork 3 children. It should use the *wait()* system call to wait for all three children to terminate. Pipe 1 should be used for communication between c1 and c2. Pipe 2 should be used for communication between c2 and c3

Prototypes for child functions should be identical and look like:

```
int childn(          /* n = 1, 2, 3          */
int in,             /* primary input file handle */
int out)           /* primary output file handle */
```

Child

- 1 Open an output file named *discard.1* with the `O_CREAT` and `O_TRUNC` options. Use your `reads()` function to read one line at a time from the primary input into a 4096 byte buffer. If a line contains the word **From**, use your `writes()` function to write entire line to your primary output. Otherwise write the entire line to *discard.1*
- 2 Use your `reads()` function to read one line at a time from the primary input into a 4096 byte buffer. Use your `reads()` function to read one line at a time from the primary input file into a 4096 byte buffer. If the line contains the @ character child 2 should extract the entire word containing the @. A newline character should then be appended to the word and the resulting string written to the primary output. "@" words should be extracted using the following algorithm. Proceed backward from the "@" until encountering either a ' ', a '<' or the start of the line. The character following the ' ' or '<' (in the direction of the @) is the first character of the @ word. If the start of the buffer is reached without finding ' ' or '<' then use the first character of the buffer as the start of the @ word. Proceed forward from the @ until encountering either a ' ', a '>', or a '\n'. The character preceding the ' ', '>' or '\n' is the last character of the @ word.
- 3 Open an output file named *discard.2* with the `O_CREAT` and `O_TRUNC` options. Use your `reads()` function to read one line at a time from the primary input into a 4096 byte buffer. See if it contains the word "Clemson", "clemson", or "CLEMSON". If so the line should be written to *discard.2*. If not the line should be to the primary output.

If you don't have a `reads()` or `writes()` function and want me to send them to you, I will do so if you ask --- needless to say, you can't submit what I send you for assignment SP1

You must use an incremental development technique.

- 1 - A two child version that duplicates stdin on stdout.
- 2 - A three child version that duplicates stdin on stdout.
- 3 - A standalone single-process test of EACH child that verifies that it works correctly. What you want to do here is use "dummy" mains that look like:

```
#ifdef TEST_CHILD1
main()
{
    child1(0, 1);
}
#endif
#ifdef TEST_CHILD2
main()
{
    child2(0, 1);
}
#endif
#ifdef TEST_CHILD3
main()
{
    child3(0, 1);
}
#endif
#ifdef THE_REAL_DEAL
main()
{
    :
}
#endif
```

The test versions involve NO FORKING AT ALL and are thus more easily tested and debugged. You can build test executables are called m1, m2, and m3 by compiling as follows:

```
gcc -o m1 -DTEST_CHILD1 -g mp1.c
```

```
gcc -o m2 -DTEST_CHILD2 -g mp1.c
```

```
gcc -o m3 -DTEST_CHILD3 -g mp1.c
```

How to submit your program:

NOTE: This procedure has **NOTHING** in common with "handin" nor "sendlab"
Do **NOT** even **TRY** to think about how they fit into this procedure because
THEY DON'T!!

<<<Do NOT turn in any image files, core files, makefiles etc.>>>

You must turn in 1 file: mp1.c

1. From any departmental Solaris system *ssh* to workstation *jmw*
2. The submission directories lie in the directory `/local/jmw2/322/mp1` which is available **ONLY IF YOU HAVE LOGGED INTO WORKSTATION jmw**. Each student has a subdirectory of `/local/jmw2/322/mp1`. The name of your subdirectory is your userid (in the example we will assume your id is *wjsmith*).
3. copy (via the `cp` command) required file to your subdirectory in `/local/jmw2/322/mp1`

For example:

```
cp /home/wjsmith/322/mp1/mp1*.c /local/jmw2/322/mp1/wjsmith
```

Here you would (hopefully) obviously need to replace
`/home/wjsmith/322/mp1/mp1.c`
with wherever you have your program.

4. Don't modify the permissions on your subdirectory. They are set so that **ONLY** you can access your directory.

After you think you have turned your programs in, its a good idea to
`cd /local/jmw2/322/mp1/wjsmith`
and make sure your files are there and they still compile and work correctly.