CSI 3131 Assignment 1 Winter 2018

(Due Date: February 8, 2017)

Process Creation and Inter-Process Communication

Goal

Practice process creation and communication in Linux using fork() and exec() system calls.

Description

Please read the complete assignment document carefully before starting.

Write a C program **dp** (standing for double pipe) with the following command syntax:

dp program1 <arglist1> : program2 <arglist2> : program3 <arglist3>

where

- program1, program2 and program3 are executable program files
- arglist1, arglist2 and arglist3 are the command line arguments of the corresponding programs
    - you may assume that the arguments do not contain characters " and :
- programk <arglistk> means execution of programk with the command line arguments from arglistk, for a given k from {1,2,3}

Your program **dp** shall launch programk <arglistk> for k=1..3. (i.e. launch three programs, each with its command line arguments)

In addition, the following requirements must be fulfilled:

1. program2 <arglist2> is executed with the standard input coming from the standard output of program1 <arglist1>
2. program3 <arglist3> is executed with the standard input coming from the standard output of program1 <arglist1>
3. program2 and program3 do not divide the output of program1 among themselves, they both get the full output of program1

Examples:

4. **dp ls -l : grep txt : grep doc**
   - launches **ls –l**, that will list the current directory
   - the output is sent to **grep txt**, which will output only the lines containing substring ‘txt’
   - the output is sent also to **grep doc**, which will output only the lines containing substring ‘doc’
   - as a result, information about all files of the current directory containing txt or doc in their filename will be outputted (if a file contains both txt and doc, it will be outputted twice)
   - in particular, if your folder has the following files:

```
fieldlv@ubuntu:~/Desktop/ass1/test$ ls -l
total 20
-rw-rw-r-- 1 fieldlv fieldlv    0 Jan 24 19:24 a
-rw-rw-r-- 1 fieldlv fieldlv    0 Jan 24 19:24 b
-rwxrwxr-x 1 fieldlv fieldlv 7840 Jan 24 19:16 dp
-rw-rw-r-- 1 fieldlv fieldlv  232 Jan 24 19:47 log.txt
-rw-rw-r-- 1 fieldlv fieldlv    0 Jan 24 19:24 t
-rw-rw-r-- 1 fieldlv fieldlv    1 Jan 24 19:23 test1.txt
-rw-rw-r-- 1 fieldlv fieldlv    1 Jan 24 19:24 test2.txt
-rw-rw-r-- 1 fieldlv fieldlv    0 Jan 24 19:26 test4.doc
```

   - then your program should get the following outputs:

```
fieldlv@ubuntu:~/Desktop/ass1/test$ dp ls -l  : grep txt : grep doc
fieldlv@ubuntu:~/Desktop/ass1/test$ -rw-rw-r-- 1 fieldlv fieldlv  232 Jan 24 19:47 log.txt
-rw-rw-r-- 1 fieldlv fieldlv    1 Jan 24 19:23 test1.txt
-rw-rw-r-- 1 fieldlv fieldlv    1 Jan 24 19:24 test2.txt
-rw-rw-r-- 1 fieldlv fieldlv    0 Jan 24 19:26 test4.doc
```

5. **dp cat /proc/cpuinfo : grep cpu : grep bug**
   - the result should be similar to:

```
fieldlv@ubuntu:~/Desktop/ass1/test$ cpu family  : 6
cpu MHz         : 2194.918
cpu cores       : 2
cpuid level     : 13
cpu family      : 6
cpu MHz         : 2194.918
cpu cores       : 2
cpuid level     : 13
fdiv_bug        : no
f00f_bug        : no
coma_bug        : no
bugs            :
fdiv_bug        : no
f00f_bug        : no
coma_bug        : no
bugs            :
```

Background information:

1. You will need the following C library functions:
    a. fork() – should be familiar from lectures
    b. pipe() – should be familiar from lectures (note, remember that multiple process can be attached to each end of the pipes, which means that a pipe is maintained until no processes are connected at either end of the pipe)
    c. execvp(const char * program, const char *args[])
        - replaces the current process with the program from the file specified in the first argument
        - the second argument is a NULL terminated array of strings representing the command line arguments
        - by convention, args[0] is the file name of the file to be executed
    d. dup2(int newfd, int oldfd) – duplicates the oldfd by the newfd and closes the oldfd. See http://mkssoftware.com/docs/man3/dup2.3.asp for more information..
    e. read(int fd, char *buff, int bufSize) – read from the file (or pipe) identified by the file descriptor fd bufSize characters into the buffer buff. Returns the number of bytes read, or -1 if error or 0 if the end of file has been reached (or the write end of the pipe has been closed and all data read).
    f. write(int fd, char *buff, int buffSize) – write into the file/pipe buffSize characters from the buffer buff
    g. close(int fd) – close an open file descriptor

2. Each process has by default three open file descriptors:
    a. Standard input (file descriptor 0, i.e. read(0,buf, 4) reads 4 characters from the standard input to the buffer buf)
    b. Standard output (file descriptor 1)
    c. Standard error (file descriptor 2)
    d. When a command is run from the shell, the standard input, standard ouput and standard error are connected to the shell tty (terminal). So reading the standard input reads from the keyboard and writing to the standard ouput or standard error writes to the display.
    e. From the shell it is possible to connect the standard output from one process to the standard output to another process using the pipe "|". For example, the command "who | wc" connects the standard output from the who process to the standard input of the wc process such that any data written to the standard output by the who process is written (via a pipe) to the standard input of the wc process.

To complete the assignment:

1. Start with the provided file dp.c. It includes all the header files you need, as well as the code for parsing the command line arguments.
2. You need to enter the name and student ID in the description area of the file dp.c
3. Implement the procedure doublePipe().
4. Submit the resulting file dp.c (DO NOT COMPRESS IT)