Computer Science and Engineering & Information Technology (2^{nd} Year B.Tech.) IIIT Kalyani, West Bengal

Operating System Lab (CS 411): (Spring: 2019-2020)

Assignment - 4 *Marks: 20*
Assignment Out: *7th February, 2020*

1. Write a C++ program (`disp.c++` $\Rightarrow$ `disp`) that will do the following:

   (a) The program takes command line arguments. The arguments are the name/path of another executable (ex) and its command line arguments. As an example `$ ./disp ls -l`, where "ex" is "ls" and `-l` is the command line argument of `ls`. The executable "ex" should write its output in `stdout`.

   (b) The program creates an unnamed pipe with file descriptors `fd[0]` and `fd[1]`.

   (c) Then the program forks two child processes c_1 and c_2. The pipe is shared by both of them.

   (d) In the child process c_1 the file descriptor for `stdout` is closed and `fd[1]` is copied to that slot. So c_1 writes in the pipe using `cout`. Then the executable "ex" is loaded in the process c_1 with its arguments. You may use `execvp()`.

   (e) In the child process c_2 the file descriptor for `stdin` is closed and `fd[0]` is copied to that slot. So c_2 reads from the pipe instead of `stdin` using `cin`. Then `/usr/bin/less` is loaded in c_2 with arguments `"-f" "/dev/stdin" NULL`[1]. You may use `execvp()`.

   (f) Your program is expected to display the output of "ex" using "less".

   Sample output:

```
$ ./disp cat temp
ufkrg;hrg;grgh
rkf;wel;kblklk;ll
52ctpu0u[[ 6h6[
 hrohphhjh[hhk
'       i
temp
io
o p5ooi5u5p u9u [
j0u 05u0vu0yum0066666666666666666
ixhhioot[ggtj
hhgeq89g5jgeqjg
igxjopegpimpg
ixppj j jjj
aaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbb
ccccccccccccccccc
ddddddddddddd
eeeeeeeeeeeee
ffffffffffff
ggggggggggggggggggg
hhhhhhhhhhhh
iiiiiiiiiiiii
jjjjjjjjjjjjjj
/dev/stdin
```

[1] `less` expects a file name, but `/dev/stdin` is not a regular file so it is necessary to 'force' it by `-f`. The `NULL` terminates the argument list for `less`.

2. Write a C++ program that does the following.

   (a) Reads two positive integer data `blockSize` and `blockCount`. It creates two buffers `buff1` and `buff2`, each of type `char` and of size `blockSize` bytes. The buffer `buff1` is initialized with `a`'s and the buffer `buff2` is initialized with `b`'s.

   (b) It creates an unnamed pipe with file descriptors `pfd[0]` (read) and `pfd[1]` (write).

   (c) It creates two child processes c_1 and c_2. The child process c_1 writes the `buff1` in the pipe (`pfd[1]`) for `blockCount` number of times using the system call `write()`. Similarly c_2 writes `buff2` in the pipe same number of times.

   (d) The parent process reads data from the pipe, one character at a time, using `cin`. It prints the number of `a`'s and the number of `b`'s from every contiguous block of `a`'s and `b`'s it reads from the pipe. As an example, if the pipe contains `aaabbbaaabbb` the output should be `a:  3, b:  3, a:  3, b:  3`.

   (e) Try with `blockSize`: 100B, 1KB, 4KB, 8KB, 64KB, and `blockCount`: 1, 2, 3 etc.

   (f) The purpose of the experiment is to see how write to an unnamed pipe is *atomic* and how it breaks down.

   Sample runs are:

```
$ ./a.out
Enter the size of block (bytes): 100
Enter the number of blocks: 1
PPID: 3783
CPID: 3784
Child (proc-1) writes 100 'a', iteration 0
CPID: 3785
Parent reads; a: 100
Child (proc-2) writes 100 'b', iteration 0
Parent reads: b: 100

$ ./a.out
Enter the size of block (bytes): 100
Enter the number of blocks: 3
PPID: 3786
CPID: 3787
Child (proc-1) writes 100 'a', iteration 0
CPID: 3788
Child (proc-1) writes 100 'a', iteration 1
Child (proc-2) writes 100 'b', iteration 0
Child (proc-2) writes 100 'b', iteration 1
Parent reads; a: 300
Child (proc-2) writes 100 'b', iteration 2
Child (proc-1) writes 100 'a', iteration 2
Parent reads: b: 300

$ ./a.out
Enter the size of block (bytes): 65536
Enter the number of blocks: 1
PPID: 3790
CPID: 3791
Child (proc-1) writes 65536 'a', iteration 0
CPID: 3792
Child (proc-2) writes 65536 'b', iteration 0
Parent reads; a: 65536
Parent reads: b: 65536

$ ./a.out
```

```
Enter the size of block (bytes): 65536
Enter the number of blocks: 3
PPID: 3793
CPID: 3795
CPID: 3796
Child (proc-1) writes 65536 'a', iteration 0
Parent reads; a: 69632
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Child (proc-1) writes 65536 'a', iteration 1
Parent reads; a: 8192
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Child (proc-2) writes 65536 'b', iteration 0
Parent reads; a: 20480
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Child (proc-1) writes 65536 'a', iteration 2
Parent reads; b: 4096
Parent reads; a: 8192
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 8192
Child (proc-2) writes 65536 'b', iteration 1
Parent reads; a: 40960
Child (proc-2) writes 65536 'b', iteration 2
Parent reads: b: 126976

$ ./a.out
Enter the size of block (bytes): 131072
Enter the number of blocks: 1
PPID: 3801
CPID: 3802
CPID: 3803
Parent reads; a: 65536
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 8192
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Child (proc-1) writes 131072 'a', iteration 0
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 4096
Parent reads; b: 4096
Parent reads; a: 20480
Child (proc-2) writes 131072 'b', iteration 0
Parent reads: b: 86016
```