CIT 483/583 Lab – Advanced File I/O & Non-Blocking I/O

Fall 2018

**Instructions**

Open and save this file in any MS word-compatible format as Lab05\_*Firstname*\_*Lastname*.<ext> and place your answers in that document. Do a **Save-As** and retain all of my content. Keep the document safe in case your submission fails, or you discover an error prior to the due date and wish to re-submit. Submit your document to the Lab05 dropbox in Canvas. The due date and any other pertinent information are noted in the Canvas item.

This lab should be completed on students.cs.nku.edu as a reference implementation and as a means of testing your answers. I have supplied the .rb file for this lab.

**Place the results of the command(s) or other answers in or immediately following each question and make sure your answers stand out from the questions by using a different font color.**

1. Copy the file write\_file.rb from my home directory on stduents.cs.nku.edu to a location somewhere under your home directory and make sure it is executable. Show where you copied it and it's full listing.

$ pwd

/home/NKU/mccordt/CIT483

$ ls -l write\_file.rb

-rwxr-xr-x 1 mccordt domain^users 649 Oct 11 11:39 write\_file.rb

1. Pertaining to the script write\_file.rb:

1. What is the significance of the trailing "i" in the expression when /^-d\d$/i in the case statement?

**It is to allow a case-insensitive match for the reg exp.**

1. In the code File.open(out\_file, mode || "w"), what does the || mean, or what effect does it have?

**If mode is not nil or is false, "w" is used as the default.**

1. Execute it with an invalid argument string and an option that causes an IOError and copy the output of both into the space provided:

(a)

./write\_file.rb –bad

./write\_file.rb:18:in `block in <main>': Unknown option: –bad (RuntimeError)

from ./write\_file.rb:4:in `each'

from ./write\_file.rb:4:in `<main>'

(b)

./write\_file.rb -r

./write\_file.rb:25:in `write': not opened for writing (IOError)

from ./write\_file.rb:25:in `puts'

from ./write\_file.rb:25:in `block (2 levels) in <main>'

from ./write\_file.rb:23:in `times'

from ./write\_file.rb:23:in `block in <main>'

from ./write\_file.rb:21:in `open'

from ./write\_file.rb:21:in `<main>'

1. In the next few exercises, we will run the write\_file.rb script in various ways to observe how Linux handles multiple processes simultaneously modifying the same file. In each of these examples, two processes will be created to run the script. When both processes are all complete, the file's contents will be displayed. You can copy/paste them into your terminal as one line. Capture the output of each step in the space provided.
2. Run the Bash command string below 2 times. It may or may not mix the output from the two processes in the console or in the test.out file (i.e., some lines may be from one process, some may from the other process) and capture both runs :

./write\_file.rb& ./write\_file.rb& wait; echo "RESULT:"; cat test.out

[1] 17812

[2] 17813

PID 17813 -- 0PID 17812 -- 0

PID 17812 -- 1PID 17813 -- 1

PID 17813 -- 2PID 17812 -- 2

PID 17813 -- 3PID 17812 -- 3

[1]- Done ./write\_file.rb

RESULT:

PID 17812 -- 0

**PID 17813 -- 1**

PID 17812 -- 2

PID 17812 -- 3

1. Similarly, run the next command 2 times. You may see several ways in which the output can be "confused". The reason for the oddities involves the fact that each process has a different "cursor", and is writing into the file at the position pointed to by its cursor. Since the two processes are writing lines of differing lengths by virtue of the –c option, when mixed, they often overwrite the other's newline characters, causing the RESULTing output to be mixed. Capture an example that has garbled output in test.out:

./write\_file.rb -c3& ./write\_file.rb& wait; echo "RESULT:"; cat test.out

[1] 17648

[2] 17649

"-c3"

PID 17649 -- 0

PID 17648 -- 0

PID 17649 -- 1

PID 17648 -- 1

PID 17649 -- 2PID 17648 -- 2

PID 17649 -- 3PID 17648 -- 3

[1]- Done ./write\_file.rb -c3

[2]+ Done ./write\_file.rb

RESULT:

PID 17648 -- 0 PIDPID 17648 --PID 17PID 17648PID 17649PID 17648 -- 3

1. This command again has both processes appending to the file. In lieu of locking, this is by far

the safest way to have multiple processes writing to the same file. Capture any example of the output of this command:

rm -f test.out; ./write\_file.rb -a& ./write\_file.rb -a& wait; echo "RESULT:" ; cat test.out

[1] 17607

[2] 17608

"-a"

PID 17607 -- 0

"-a"

PID 17608 -- 0

PID 17607 -- 1

PID 17608 -- 1

PID 17607 -- 2

PID 17608 -- 2

PID 17607 -- 3

PID 17608 -- 3

[1]- Done ./write\_file.rb -a

[2]+ Done ./write\_file.rb -a

RESULT:

PID 17607 -- 0

PID 17608 -- 0

PID 17607 -- 1

PID 17608 -- 1

PID 17607 -- 2

PID 17608 -- 2

PID 17607 -- 3

PID 17608 -- 3

1. In this example, both processes acquire the lock before writing. (You should see one process finish, before the other writes its first line. (The last process in "wins".) Capture any example of the output of this command:

./write\_file.rb -L& ./write\_file.rb -L& wait; echo "RESULT:"; cat test.out

[1] 12400

[2] 12401

PID 12400 --0

PID 12400 --1

PID 12400 --2

PID 12400 --3

PID 12401 --0

[1]- Done ./write\_file.rb -L

PID 12401 --1

PID 12401 --2

PID 12401 --3

[2]+ Done ./write\_file.rb -L

RESULT:

PID 12401 --0

PID 12401 --1

PID 12401 --2

PID 12401 --3

1. In this example, both processes open in "append" mode and acquire the lock. Thus, they take

turns writing to the file. Capture any example of the output of this command:

rm -f test.out; ./write\_file.rb -a& ./write\_file.rb -a& wait; echo "RESULT:"; cat test.out

**[1] 12801**

**[2] 12802**

**PID 12801 --0PID 12802 --0**

**PID 12802 --1PID 12801 --1**

**PID 12801 --2PID 12802 --2**

**PID 12802 --3PID 12801 --3**

**[1]- Done ./write\_file.rb -a**

**RESULT:**

**PID 12801 --0**

**PID 12802 --0**

**PID 12802 --1**

**PID 12801 --1**

**PID 12801 --2**

**PID 12802 --2**

**PID 12802 --3**

**PID 12801 --3**

1. In this case, only one process acquires the lock and the other ignores it. The output should be similar to the preceding case in which neither process acquired the lock. (Locks are ineffective if even one process ignores it.) Note that the –I option is used to change the number lines of output from one process. Capture any example of the output of this command:

rm -f test.out; ./write\_file.rb -i6 -a -L& ./write\_file.rb -a& wait; echo "RESULT:"; cat test.out

[1] 12809

[2] 12810

PID 12809 --0

PID 12810 --0

PID 12809 --1PID 12810 --1

PID 12810 --2PID 12809 --2

PID 12810 --3PID 12809 --3

PID 12809 --4

PID 12809 --5

[1]- Done ./write\_file.rb -i6 -a -L

[2]+ Done ./write\_file.rb -a

RESULT:

PID 12809 --0

PID 12810 --0

PID 12809 --1

PID 12810 --1

PID 12810 --2

PID 12809 --2

PID 12810 --3

PID 12809 --3

PID 12809 --4

PID 12809 --5

1. Run the following commands, provide the output, and briefly explain what the output after the word "Found:" represents:

ruby -e 'IO.popen("wc", "r+"){|p| p.puts "hello"; p.close\_write; puts "Found:

#{p.read}"}'

> #{p.read}"}'

Found:

1 1 6

1 1 6 is the output of wc when applied to the string "hello."

ruby -e 'IO.popen("cat", "r+"){|p| p.puts "hello"; p.close\_write; puts "Found: #{p.read}"}'

Found: hello

"hello" is the result of the cat command.

ruby -e 'IO.popen("ls -l", "r"){|p| puts "Found: #{p.read}"}'

Found: total 16

-rw-r--r-- 1 mccordt domain^users 140 Oct 18 14:21 test.out

-rwxr-xr-x 1 mccordt domain^users 656 Oct 18 14:10 write\_file.rb

-rwxr-xr-x 1 mccordt domain^users 795 Oct 18 13:11 write\_file\_2.rb

-rw-r--r-- 1 mccordt domain^users 56 Oct 18 12:51 write\_file.out

The child process receives the output of the ls –l command.