**CSE-381: Systems 2**

**Exercise #4**

Max Points: 15

|  |
| --- |
| **Prior to proceeding with the exercise, you should first save/rename this document using the naming convention MUid\_Exercise4.docx (example: raodm\_Exercise4.docx).**  **Objective**: The objective of this part of the exercise is to:   * Review the operations of fork system call. * Run different programs using fork and execvp/execlp system calls. * Review object-oriented programming in C++   **Submission**: Once you have completed this exercise, upload:   * This MS-Word document saved as a PDF file with the naming convention **MUid\_Exercise4.pdf**. * The C++ source files you modified in the different parts of this exercise.     You may discuss the questions with your neighbors, TAs, or your instructor. |

|  |  |
| --- | --- |
| **Name:** |  |

|  |  |
| --- | --- |
| Video camera | It may be helpful to recollect concepts regarding execlp and execvp system calls by reviewing the following video <https://youtu.be/7SZdWV1fLxA?t=582> (**starting at the 9:40 minute mark**) to recap the concepts associated with the exec family of syscalls used in this part of the exercise. Once you have reviewed the video then proceed with this exercise. |

# Part #1: Understanding the basics of execvp and execlp syscalls

**Background**: The task of running a completely different program is performed by the execvp system call (The execvp system call is one in a family of exec system calls). The execvp system call requires the following 2 arguments:

1. The first argument is the name or path of the program to be executed
2. An array of C-strings for command-line arguments. Note: As per convention, the 1st command-line argument should always be exactly the same as the name/path of the program being executed.

**Exercise**: In this part of the exercise, we are going to build some strength on the basic understanding of the execvp system call:

1. What should be the first command-line argument passed as part of execvp system call? why?

|  |
| --- |
|  |

1. Upon successfully running a given program, what is the return value of the exec system call?

|  |
| --- |
|  |

1. Why do we need to typically fork before using exec?

|  |
| --- |
|  |

**Exit code questions**

1. What is exit code?

|  |
| --- |
|  |

1. How is the exit code value meant to be interpreted?

|  |
| --- |
|  |

1. What system call can a parent process use to obtain the exit code of a child process? **Show an example code fragment**.

|  |
| --- |
|  |

**Question on myExec helper method**

1. What was the signature (name and parameters) of the helper method that was suggested to ease using the execvp system call (Hint: see: 16 minutes into the video)? What was the caution discussed about the helper method?

|  |
| --- |
|  |

# Part #2: Programming with execvp method with C++ classes

**Background**: Object oriented programming provides a convenient approach to encapsulate information and provide controlled access to it. As a convention in C++, we use a split design for classes –

* The header (.h) file: The API of the class is defined in a header file (so it can be included in other source files).
* The source (.cpp) file: The methods in the class are implemented a separate source file.

**Exercise**: In this exercise you will be completing a given C++ class to streamline forking and running programs.

1. Start NetBeans and create a new Miami University C++ Project , without a main file, named exercise4 on your GNU/Linux VM.
2. Download the starter code, named ChildProcess.h, ChlidProcess.cpp, serial\_vs\_parallel.cpp, main.cpp, and data.txt to your exercise4 project directory.
3. Add all 4 files to your project. **Note**: the starter code it will not compile yet as some methods are not yet implemented.
4. Briefly review the methods declared in ChildProcess.h.
5. Using the Javadoc/doxygen style comments in the header file, implement the methods in the source (ChildProcess.cpp) file. **Note**: You don’t need to worry about serial\_vs\_parallel.cpp in this part.
6. Once you have correctly implemented the methods in ChildProcess.cpp, you will be able to run the program.
7. **Expected output from the program** (1 successful command with zero exit code and 1 unsuccessful program with non-zero exit code) is shown below:

|  |
| --- |
| \_\_ \_\_ \_\_\_\_\_ \_  \ \ / / | \_\_ \ | |  \ \\_/ /\_\_ \_ \_ | |\_\_) |\_\_\_ \_\_\_| | \_\_  \ / \_ \| | | | | \_ // \_ \ / \_\_| |/ /  | | (\_) | |\_| | | | \ \ (\_) | (\_\_| <  |\_|\\_\_\_/ \\_\_,\_| |\_| \\_\\_\_\_/ \\_\_\_|\_|\\_\  **Exit code: 0**  ls: cannot access '/blah': No such file or directory  **Exit code: 512**  Calling runProcesses()...  runProcesses() completed. |

1. Let’s ensure we are paying attention to API of the int ChildProcess:wait() const method
   1. What does the const at the end of this method indicate – as in what can/cannot this method do?

|  |
| --- |
|  |

* 1. Show an example of a C++ statement that would be invalid in this wait method.

|  |
| --- |
|  |

# Part #3: Understanding serial vs. parallel execution

**Background**: In multiprocessing operating systems multiple processes can run simultaneously to take full advantage of the multi-core computers. Typically, a child process is started and we wait for the process to finish. This mode of operation is called serial or sequential processing, where only 1 child process runs at time. On the other hand, multiple processes can be run by first starting up several child processes (one after another) and then waiting for them to finish. This mode of operation is called parallel processing. Given sufficient hardware, parallel processing enables performing many tasks simultaneously. However, care must be taken so as not to overload the machine – too many processes can consume memory/resources and increase context-switching overheads, thereby negatively impacting overall performance.

**Exercise**: In this part of the exercise you are expected to covert a given serial program to run in parallel, via the following procedure.

1. Copy-paste the following starter code into the body of the runProcesses method in the serial\_vs\_parallel.cpp source file.

|  |
| --- |
| **void** **runProcesses**() {  // The command to be run -- this just sleeps for 5 seconds.  **const** StrVec cmd = {"sleep", "5"};  // Run the same process 3 times in serial fashion.  ChildProcess cp;  // 1st run  cp.forkNexec(cmd);  cp.wait();  // 2nd run  cp.forkNexec(cmd);  cp.wait();  // 3rd run  cp.forkNexec(cmd);  cp.wait();  } |

1. Compile and run your program. You will notice that the program takes about 15 seconds to run. Copy-paste the output from the NetBeans terminal showing the runtime in the space below:

|  |
| --- |
|  |

1. Let’s see which lines of code take time to run by stepping through the debugger –
   1. Set a breakpoint at the 2nd line of the runProcesses method with the first call to the forkNexec method.
   2. Step through the code and observe which line of code (or method call) takes most time to run. Briefly describe why you think this line of code or method call takes so much time to finish?

|  |
| --- |
|  |

* 1. Repeat stepping through the debugger and make a screenshot of the debugger showing the line of code that takes most time to run. Place screenshot below:

|  |
| --- |
| Place screenshot of debugger (as instructed above) in this space. |

1. Now modify the program to run the 3 processes in parallel. **Tip**: You will need 3 ChildProcess objects, 1 for each process.
2. Upon correctly converting to parallel operation, the program should run in 5 seconds.
3. Now, repeat the debugging operation to step through the code again to observe which line of code takes the most time
   1. Indicate which line of code (or method call) takes most time

|  |
| --- |
|  |

* 1. If the above method call takes time, then how come subsequent calls to the same method finish quickly?

|  |
| --- |
|  |

1. Next, swap/change the order of calls to waitpid method in the revised parallel version (that is, wait for child 3, then child 2, then child 1). Does the runtime change? Why-or-why not?

|  |
| --- |
|  |

1. Now, suppose that the serial version had 3 processes running 3 different commands, namely: sleep 5, sleep 4, and sleep 3. Then

|  |  |
| --- | --- |
| Runtime of the **serial** version with the 3 processes would be |  |
|  |  |
| Runtime of the **parallel** version with the 3 processes would be |  |

# Part #5: Practice question from exam(s)

*Estimated time: 20 minutes*

**Background**: Expect programming questions in exams that are based on the concepts covered in labs and homework. Here is a practice problem from exams related to fork & exec.

|  |
| --- |
| **Example output:** |
| $ ./ex1 23 5741 18 3347  23: terminated  5741: not terminated  18: not terminated  3347: terminated |

**Exam question**: Given a pid as command-line argument, the kill Linux command terminates the given process with exit code 0 (zero) if the process was terminated (and non-zero exit code otherwise to indicate invalid pid or permission denied etc.). Complete the program below to run the kill command to terminate a list of processes whose pids are specified as command-line arguments, and print results, as shown in the example output.

// Assume all #include s are here!

**int** **main**(**int** argc, **char** \*argv[]) {

|  |
| --- |
|  |

**return** 0;

}

# Part #4: Submit files to Canvas

*Estimated time: < 5 minutes*

Upload just the following files to Canvas via the CODE plug-in:

* The two C++ source files that you have modified (*i.e*., ChildProcess.cpp, and serial\_vs\_parallel.cpp).
* Upload this document (duly filled with the necessary information) saved as a PDF file using the naming convention MUid\_Exercise4.pdf.
  + **Note**: If your PDF is large, then the CODE plug-in may not accept it. In this case, complete submission of your C++ source files. After you have completed your submission, then attach the PDF as a submission-comment to your submission.
* Ensure you actually finish the submission process!

Upload each file individually onto Canvas. Do not upload zip/7zip/tar/gz or other archive file formats for your submission.