Fork Execvp

This assignment requires designing forks to create several child processes. One-way pipes are used to communicate between the parent process and each child process. Each child process will use information sent by the parent over a pipe to execute the execvp(). The execvp() will overlay each child using execvp() to execute calculate.

The main program will also use fork to display using the bash cat command in forked children processes to display the answer files.

This assignment will specify the requirements for the programs.

General Programming Requirements

This assignment requires the development of 2 cpp files and associated execution files:

- calculate
  - calculate.cpp
  - calculate
- forkexecvp
  - forkexecvp.cpp
  - forkexecvp

Develop both programs and associated execution files in the same development directory.

The following rules apply to all programs used in this assignment.

Throughout all programs, use LineInfo.h exception handling for checking that all operations succeed.

Use global constants in all the programs.

Do not use FILE and the f_ file commands for file operations.
Use the open(), read(), write(), close()… for file operations.
So, there is not any file descriptor usage in this assignment.
Use C++ streams capability instead.

Do use all modern C++ (g++) features (iostream, file streams, string, … etc.).

Build using g++ -std=c++11 as described in the assignment guides to use the correct version 11 of the compiler to get the C++ modern constructs accepted.

It is strongly recommended that for this assignment to develop the calculate program first. Since it is an easier program to design. Also, the calculate program can be independently tested by running it with arguments from the bash shell.

The assignment requires that both the forkexecvp and calculate are operational to run the assignment.

Fork Execvp

calculate Program

Program Command Line

The running **calculate** program will accept from the command line the following arguments:

```
calculate < sum | average | greatest | least > filename.txt
```

filename.txt – a text file that contains integers, 1 integer per line

The programmer can create their own file filled with random integers.

Example:

```
./calculate sum filerandom.txt
```

Use the following includes to use the STL vector class.

```
#include <vector>
#include <iterator>
#include <numeric>
#include <algorithm>
```

Define a correct main function using (argc, argv).

Use the LineInfo.h exception handler to check if the number of arguments to calculate are correct.

If the command line is not correct,
the following shall be displayed:

```
EXCEPTION:
Oh my Goodness...


Wrong arguments
Usage:
calculate < sum | average | greatest | least > filename.txt
filename ▒ a file that contains integers, 1 integer per line Example:
calculate sum filerandom.txt

in calculate.cpp",line:25

Press the enter key once or twice to leave...
```

If the command line has an invalid command, the following is to be displayed:

```
EXCEPTION:
Oh my Goodness...


Invalid Command: command
Usage:
calculate < sum | average | greatest | least > filename.txt
filename ▒ a file that contains integers, 1 integer per line Example:
calculate sum filerandom.txt

in calculate.cpp",line:65


Press the enter key once or twice to leave...
```

Fork Execvp

Define a vector of int type for use in this program.

Define an ifstream text file object using the argv[2] as the file name.

Define an ifstream istream_iterator for int types for the above istream object.

Use the copy function with the above definitions to copy the file stream content to the vector.

The dynamic vector will automatically grow to assign the file stream integers into the vector.

Parse on the argv [1] command to execute the following commands:
< sum | average | greatest | least >

Check for invalid commands using the LineInfo.h exception handler.

Use the following vector methods to calculate the command:
accumulate()
size()
min_element()
max_element()

When program is correctly done executing return EXIT_SUCCESS.

Test the program with a small random number file that you created and using the various commands.

Once this is all done, calculate will be ready for your usage in the forkexecvp execution program via the execvp() function.

Fork Execvp

forkexecvp Program

Program Command Line

This program will accept the following arguments into argv[]:

```
forkexecvp <file to create> <amount of random numbers> <random number range>
```

random number range : 1 to random number range inclusive.

Example:

```
./forkexecvp randomnumber.txt 10000 100
```

Use a string object to hold the random file name in from the command line stored in argv[1]. Use atoi() on argv[2] and argv[3] to hold the number of random numbers and the range, respectively.

Using these 3 definitions as arguments develop a function to write out the defined number of random numbers in the random number range to the defined file name.

Use:

```
    time_t t; srand((unsigned)time(&t));
```

To seed the random number generator formula.

Define an array to be used as a one-way communication pipe.

Define a string message array that contains the 4 string messages:
"sum", "average", "greatest", "least"

For Loop:

Define a master for loop that will execute the loop for each child.

Inside the loop:

Create a pipe using the array defined above.

Execute a fork();

Determine which process you are in by the fork() return value.

Parent Process Code:

Close the unnecessary READ pipe.

Display a status message - see the activity run at the end.

Execute a write to the WRITE pipe, using the string messages array with an index that is the control variable in the for loop and using the size of command on the string array.

Note - pipe writes do not take as arguments string types, so use the .c_str on all string arrays inside the argument list to convert to classic c-strings.

Fork Execvp

Child Process Code

Close the unnecessary Write pipe.

Define a clear (= {0}) char array for to be used in the pipe read with the correct size declarator.

Read from the READ pipe using the pipe read array.

Display a status message (see activity run at the end).

Prepare an argument list for execvp.

```
char * arglist[] = { (char *)"./calculate", pipeReadMessage,
                     (char*)randomFileNameStr.c_str(), NULL };
```

Note – the argument list does not take as elements string types, so use the .c_str on all string objects in the list to convert the strings to classic c-strings.

Note – the classic c-strings must end with a NULL, the .c_str will provide the c-string ending null.

Display a status message (see activity run at the end of this document).

Replace stdout stream for the answer file named with pipeReadMessage.

```
string answerFileName = "answer";
answerFileName += pipeReadMessage;
answerFileName += ".txt";
```

Execute the fstream redirection operator to replace stdout with the file for writing.

```
freopen(answerFileName.c_str(), "w", stdout);
```

Note – fropen does not take as arguments string types, so use the .c_str on all string types inside the argument list to convert to classic c-strings.

From this point on in the execvp created process calculate will send to the stream file object all that are going to cout (cout <<).

Close the read stream type because the calculate program is done using it.

Execute the execvp function using the argument list [0] and the previously defined argument list.

Now a new calculate image execution file has overlaid (replaced) the child process.

All code after this execvp invocation will now not execute.

The only time there is a return is when the execvp invocation operation fails, so use the LineInfo.h exception handler to check for this.

Once the for loop is done:

Display a status message (see activity run at the end of this document).

Wait for all children to end.

Display a status message (see activity run at the end of this document).

Close the write pipe.

Fork Execvp

Display Answer Files from Within the Program

Display a status message (see activity run at the end of this document).

Define a master for loop that will execute the loop for each child type.
Reuse the previous for loop.

Inside the loop:

Execute a forK();

Determine which process you are in by using the fork() return value.

Only check to see if you are in the child process, do not with the parent process.

Child Process

Construct an answer file name as a sting using the string array messages with the index being the loop control variable.

Display a status message (see activity run at the end of this document).

Create the following argument list for the execvp():

```
char* arglist[] = { (char*)"cat", (char*)answerFileName.c_str(), NULL };
```

Execute the execvp function using the argument list [0] and the previously defined argument list.

Once the for loop is done:

Display a status message (see activity run at the end of this document).

Wait for all children to end.

Display a status message (see activity run at the end of this document).

Fork Execvp

Activities File

Run the forkexecvp program with the following arguments:

```
./forkexecvp filerandom.txt 100 100
```

Use the copy and paste method to copy to a local text file named **forkexecvpactivities.txt** the program run and results. Do not use script or terminal session or you will lose points.

See an example activity result at the end of this document.

Submittal

Put in a zip folder that you name whatever you want:

Forkexecvp.cpp
calculate.cpp
forkexecvpactivities.txt (the activity file)

Submit whatever you called the .zip folder to blackboard.

Fork Execvp

```
{cs1:~/courses/3377/assigns/04} ./forkexecvp randomnumber.txt 100 100

./forkexecvp randomnumber.txt 100 100
Parent pid : 44626 to Child Process No : 0
Send Message : sum
Parent pid : 44626 to Child Process No : 1
Send Message : average
Child pid : 44627 Child Process No : 0 Received Message : sum
Child pid : 44627 Child Process No : 0
execvp(./calculate, ./calculate, randomnumber.txt, NULL)
Parent pid : 44626 to Child Process No : 2
Send Message : greatest
Child pid : 44628 Child Process No : 1 Received Message : average
Child pid : 44628 Child Process No : 1
execvp(./calculate, ./calculate, randomnumber.txt, NULL)
Child pid : 44629 Child Process No : 2 Received Message : greatest
Parent pid : 44626 to Child Process No : 3
Child pid : 44629 Child Process No : 2
Send Message : least
Parent pid: 44626 Start - Wait for all calculate children to finish
execvp(./calculate, ./calculate, randomnumber.txt, NULL)
Child pid : 44630 Child Process No : 3 Received Message : least
Child pid : 44630 Child Process No : 3
execvp (./calculate, ./calculate, randomnumber.txt, NULL)
Parent pid: 44626 Done  - Wait for all calculate children to finish

Parent pid : 44626 Use execvp() cat to display answer files:
Child pid : 44632
cat answersum.txt
Parent pid: 44626 Start - Wait for all display answer file children to finish
Child pid : 44633
cat answeraverage.txt
Child pid : 44634
cat answergreatest.txt
Child pid : 44635
cat answerleast.txt
./calculate sum randomnumber.txt
Sum      : 4967
./calculate average randomnumber.txt
Average  : 49.67
./calculate least randomnumber.txt
Least    : 3
./calculate greatest randomnumber.txt
Greatest : 100
Parent pid: 44626 Done  - Wait for all display answer file children to finish
```