Assignment 5

COMP 2401

Date: November 26, 2017

Due: on December 8 2017 before 23:55

Submission: Electronic submission on cuLearn.

Objectives:

a. Reading from binary files: opening and closing a file (fopen() and fclose(), reading from a file (fread())

b. Process spawning: differentiating between parent and child processes (fork()), examining the return code of the child process(), waiting for a child process (wait() and waitpid())

c. Program Morphing – changing from one program to another (excevp())

d. Inter process communication – using signals and return code

e. Usage of signals – signals and function pointers.

Submission (10 pts.)

- Submission must be in cuLearn by the due date.
- Submit a single tar file with all the c and h files.
- Submit a Readme.txt file explaining
    - Purpose of software
    - Who the developer is and the development date
    - How the software is organized (partitioned into files)
    - Instruction on how to compile the program (give an example)
    - Any issues/limitations problem that the user must be aware of
    - Instructions explaining how to use the software

Coding Instructions:

1. Comments in Code – as provided in the slides given in class
2. No usage of global variables. All data must be passed or received via function parameters except when specifically stated.

Provided Resources

The following files are provided in A4.tar

1. isPrime.c – the base program for checking whether a number is a prime number
2. prime.bin – a file that contains 12 unsigned numbers in binary format
3. prime.txt – a file that contains 12 unsigned numbers in ASCII format. The numbers correspond to the numbers in the file prime.bin
4. createBinary.c – a program that creates a binary file from a given set of numbers. Assuming that the executable is a.out the usage is: a.out filename num1 num2 etc. where filename is the binary file name, num1 is the first unsigned number num2 is the second unsigned number etc.

Grading:

Assignment is graded out of 100

You will be graded with respect to what you have submitted. Namely, you earn points for working code and functionality.

Grading will look at how the functions are coded, whether functions meet the required functionality, memory allocation and de-allocation, setting pointers to NULL as needed, etc.

1 Background

You are tasked to write a program that accepts a set of unsigned integers and determines which numbers are prime number. In order to complete the task you are given a program (isPrime.c), which accepts a single unsigned integer as a command line parameter and return:

0 if the number is not a prime number

1 if the number is a prime number

2 if the command line argument did not include a number

The given program assumes that the input, if provided to the program, is correct and therefore the program does not require to check whether the input is an unsigned integer.

Not all programs can fully utilize the computation power of the cpu. Often a program is required to wait for slow resources (e.g., reading from or writing to a disk).

The assignment mimics such programs by artificially inserting wait time during execution see function usleep(x) which makes the program sleep for x micro seconds 0.000001 seconds. In this program x=100 (0.1 milliseconds or 0.0001 seconds in each for loop). This allows the assignment to present the benefit of task parallelization or task distribution among multiple processes.

This assignment is a progressive assignment. Namely, it is divided into several tasks each building on the previous one. For each task (1-3) you will be asked to create a program (including a makefile). Thus, once you completed the code for one task you can use it (by duplicating it) and expanding it in order to complete the next task.

2 Task 0 Understanding the base code

In this task you will understand the prime testing program that was provided.

1. Review the program isPrime.c. Make sure that you understand the program and how it works. Do the following:
   1.1. Compile the program and create an executable called isPrime.
   1.2. Test the program as follows:
      1.2.1. Test 1 isPrime 1535068679
      1.2.2. Test 2 isPrime 1535068677
      1.2.3. Test 2 isPrime 39821

   Since you will not see any output use the debugger to ensure that your code is working by putting breakpoints at each of the return() function calls. Alternatively, you can **temporarily** add a print statements indicating whether the number was a prime number of note. This can assist you in testing Task I.
2. Compile the file and create an executable program called isPrime.

3 Task I Morphing (40 points)

Task overview

In this task you will write a program that will morph itself into the isPrime program. Here you will take advantage of the isPrime program from Task 0.

You can assume that if the file exists then it contains at least one unsigned integer

The program that you write will read from a binary file the first unsigned integer and then morph itself to the isPrime program. The file name will be given as a command line parameter.

To do

1. (10 pts.) Create a program singlePrime, use file singlePrime.c, that accepts the binary file name as a single command line parameter. Here you can use the tutorial tools that you developed.
   1.1. (5 pts.) The program will check that a file name was provided as command line parameter. If not file was provided it will print how to use the program – Usage: singlePrime fileName.
   1.2. (5 pts.) The program will check if the file exists. If the file does not exist then the program should produce an error message: file file.bin does not exist, where file.bin is the provided file name
2. (15 pts.) Create a function morph() which will take as input a string (the input number) and morph itself to the isPrime program using the excev or execvp system function call.
   2.1. Prototype int morph(char *number);
   2.2. Input: unsigned number to be checked
   2.3. Return - -1 if the morph failed
3. (15 pts.) If the binary file exists then The program should read the first unsigned integer from the file, convert it to a string using the function sprintf() and then call the morph function that you created.
4. Test your program. Here you are provided with: a. a binary file prime.bin which contains 12 unsigned integer in a binary format. b. a utility to create a binary file (the utility is createBinary.c).

4 Task II Spawning a single child (30 points)

Task overview

In this task you will write a program that will spawn a child process. The child process will morph itself into the isPrime program. The parent program will wait until the child program completes its task and printout whether the given input number is prime number using the return code from the child process.

To do

1. (5 pts.) Copy the program from Task I into a file with the name singleSpawn.c. If you added a print statement to the isPrime program (see Task 0) then remove it.
2. (10 pts.) Expand the program to so that the program shall spawn a single child. The child should morph itself into the isPrime program using the function from Task I
3. (10 points) The parent program should wait until the child process has completed its execution and then, using the return code from the child process, print whether the input number is a prime number or not a prime number depending on the return code from the child process.
   3.1. Here you will have to use the wait() function.
4. (5 points) Create a makefile to compile the program. Makefile name should be Makefile2. The executable program name should be singleSpawn.

5 Task III Spawning multiple children (50 points)

Task overview

In this task you will write a program that will spawn multiple child processes. Each child process will morph itself into the isPrime program. The parent program will wait until all the child processes have completed their work and then prints all the prime numbers.

You can assume that the binary file (if it is exists) contains at least 10 unsigned int numbers. For simplicity the program will need to process only the first 10 numbers. You can also obtain additional points by doing the bonus section in this task.

To do

1. (5 points) Copy the program from Task II into a file with the name multiSpawn.c.
2. Reading data from file - (10 points) The program shall read the first 10 numbers into an array of unsigned integers.
3. (10 points) The program should spawn a child process for each of the input numbers. Each child should morph itself into the isPrime program using the function in Task I.
   3.1. The program needs to spawn all child processes before it starts to collect the responses from the child processes.
4. (30 points) The parent program should only print the prime numbers in the given input. This will be done by examining the return code from the child processes.
   4.1. (5 points) Here you will have to use the waitpid() function, which waits for child processes to complete their tasks. You will invoke the function as waitpid(-1, &status, 0) where -1 indicates wait for any child process to complete their task.
   4.2. (20 points) In order to complete this step the program will need to "remember" which prime number is assigned to which child process. The simplest way of handling it is to create an array of 10 integers, e.g., childProcessIds[], and then assigning the process id of each spawned child in Step 3 to the corresponding array location. For example, assume that the numbers, which were read from the file, are stored in the array numbers[]. Thus, when the program spawns a child to process the third number (i.e., numbers[2]), then the program would store the child process id (pid) in childProcessIds[2]. Then when the waitpid() returns the child process id, cpid, the parent program can search for cpid in the array pid and prints the corresponding integer if needed.
   4.3. (5 points) The parent program also needs to know when to quit. This can be done in two ways a. check the return code from waitpid(); If it is -1 then there are no more children or an error has occurred. In this case the program can quit. B. count how many times it has received input from the children and quit once argc-1 children have responded.
5. (5 points) Create a make file for the program. Makefile name should be Makefile3. The program name should be multiSpawn
6. Bonus Section (10 points)
   6.1. Allocating memory complete Section 3 and 4 above by determining how many numbers are in the file and then allocating memory for the array of numbers and array of process ids. Determining how many numbers are contained in the binary file is accomplished by determining the file size: a. moving the file pointer to the end of the file using fseek() and then b. determining the file size using ftell().

Bonus Task IV Signals (20 points)

In this task you will write enhance the program in Task III to use a simple signal SIGUSR1

1. Add a counter to your program from Task III, which tracks/collects the number of process that have responded so far.
2. If signal SIGUSR1 is invoked then the program should print how many processes have finished their execution and how many processes are still working. Note, that in order to accomplish this you will need to use global variable within the file scope. For example you can either declare it as static int countFinished or int countFinished. You will also need to keep as a global variable the total number of child processes that were launched. Otherwise you will not be able to complete access the information from the interrupt function.
3. Create a make file for the program. Makefile name should be Makefile4. The program name should be multiSpawnSignal