Assignment 4

COMP 2401

Date: November 19, 2015

Due: on <u>December 2 2015 before 23:00 (10:00 PM)</u>

Submission: Electronic submission on cuLearn.

Objectives:

a. Process spawning

b. Program Morphing

c. Inter process communication

d. Usage of signals

e. Usage of file i/o

Submission

- 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
- Each task should have its own executable file and code. The file names should correspond to the task. Namely, for task1 the code file should be ***task1.c*** and the executable file should be ***task1***
- Submit the required Makefiles which will compile the code
-

Grading:

- You will be graded with respect to what you have submitted. Namely, you earn points for working code and functionality.
- The 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.
- Deductions will be given for
    - missing a Makefile (5 pts.)
    - missing a Readme file (5 pts.)

Background

You are tasked to write a program that detects which numbers in a given set of numbers is a prime number and print the numbers. In order to complete the task you are given a function, isPrime(int number), which accepts an integer number as input and returns

0 if the number is not a prime number

1 if the number is a prime number

Note that the function assumes that the input to the function is correct and it does not check whether the input number is a negative number.

```
int isPrime(int number)
{
	int i;
	for(i = 2; i*i < number; i++) {
		usleep(101);
		if (number % i == 0) {
			return(0);
		}
	}
	return(1);
}
```

Not all programs can 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=101 (which is about 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.

Coding Instructions:

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

Task 0 Morphing (10 points) (30 minutes)

In this task you will write a program that will determine whether a given number is a prime number.

1. (10 pts) Write a program that will accept a single positive number as a command line parameter and using the function isPrime above determines whether the number is a prime number.
2. Your program should return:
   2.1. 0 - if the number is not a prime number
   2.2. 1 – if the program is a prime number
   2.3. 2 – if the command line does not contain a number
   2.4. Compile the program and create an executable called isPrime.
   2.5. Test the program as follows **isPrime 1535068679** and **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.
3. Submit a makefile name Makefile0 which creates the executable isPrime. Five points will be deducted for not submitting the required makefile.

Task 1 Morphing (15 points) (30-90 minutes)

In this task you will write a program that will morph itself into the **isPrime** program for checking whether a given input number is a prime number.

1. (15 pts) The program should accept a positive number as a command line parameter. If the command line does not include a command line parameter then the program should return 2. It a command parameter exists then the program can assume that the command line parameter is a positive integer.
   1.1. Create a function ***morph(char *number)*** which will take as input the number (in a string format) and morph the program to the **isPrime** program using the **excev** or **execvp** system function call.
2. Test the program as follows **task1 1535068679** and **task1 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.
3. Submit a makefile name Makefile1 which creates the executable **task1**. Five points will be deducted for not submitting the required makefile.

Task II Spawning a single child (30 points) (30-90 minutes)

In this task you will implement a manager-worker paradigm by writing a program that will spawn a child process. The child process will then morph itself into the **isPrime** program. The parent program will wait until the child program would complete its task and printout whether the given input number is prime number or not using the return code from the child process.

1. (5 points) Create a program, called task2.c that accepts a single integer as a command line parameter. The program can assume that the input is correct (a positive integer) and no input checking is required.

   1.1. The program should check if the command line consists of a number. If it does not then the program should print out how to use the program and exit. Note, you can assume that the input number is a positive integer.

2. (10 points) The program shall spawn a single child. The child should morph itself into the **isPrime** program using the function **morph()** in 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.

   3.1. Here you will have to use the wait() function.

4. (5 pts.) Using the output of the wait() function you will also check if the child process was signalled to terminate its execution.

5. Submit a makefile name Makefile2 which creates the executable **task2**. Five points will be deducted for not submitting the required makefile.

Task III Spawning multiple children (65 points) (90-150 minutes)

In this task you will implement a manager-worker paradigm using multiple workers. Here 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 printout all the prime numbers.

1. (15 points) Create a program that accepts multiple inputs as a single integer as a command line parameter (for example **task3** 123 456 789 1234 57). The program can assume that each input is a positive integer (no input checking is required).

   The program should check if the command line consists of at least one number. If it does not then the program should print out how to use the program and exit.

2. (15 points) The program shall spawn a child process for each of the input numbers (note that **argc-1** child processes should be spawned). Each child should morph itself into the **isPrime** program using the function from Task I.

3. (35 points) The parent program should only print the prime numbers in the given input. This will be done using the return code from the child processes.

   3.1. (5 pts) 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 the task and status is the return code.

3.2. (10 pts) In order to complete this step the program will need to "remember" the prime number that was assigned to each child process. The simplest way of handling it is to create an array **workersPid** of **argc** integers and then for each spawned child spawn in step 2 assign the child pid to the corresponding array location. For example, assuming that the input consists of four integers that must be checked then when the program spawns a child with the number in argv[3] which is the third integer then the program will record the child process id in **workersPid[3]** as **workersPid[3] = cpid**. 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.

3.3. (10 pts) 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.

3.4. (5pts) The program shall only print the prime numbers to the screen

3.5. (5 pts.) Using the output of the wait() function you will also check if the child process was signalled to terminate its execution.

4. Submit a makefile name Makefile3 which creates the executable **task3**. Five points will be deducted for not submitting the required makefile.

Bonus Task IV Signals (20 points) (30-60 minutes)

In this task you will write enhance the program in Task III to use a simple signal SIGUSR1. Namely, all requirements of Task III must be completed.

1. (5) Add a counter to your program from Task III, which tracks collects the number of process that have responded so far.
2. (15) 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. Submit a makefile name Makefile4 which creates the executable **task4**. Note that this task will not be graded unless you submit the require makefile (Makefile4).

Bonus Task V reading from a file (30 points) (60-90 minutes)

In this task you will write enhance the program in Task III or Task IV. Namely, all requirements of Task III must be completed. Here you will use file I/O to read the input and provide output.

1. (25) The input will consist of a binary file that stores long integer number in it. The main program will receive in the command line the name of the input file, and the record locations to be examined. For example, the program will be invoked using the following command: task5 num.bin 1 2 -4 -8 34.

   1.1. (5 pts) Using command line properly

   1.2. Using input file correclty

      1.2.1. Input file – the input file is a binary file. The file will consist of a large number of long integers (all positive numbers).

      1.2.2. (25) Records number to be examined – the record numbers are either positive or negative numbers. If the record number is positive then it means read the record relevant to the beginning of the file. For example if record number is 1 then the number to be examined is the first record in the file. Similarly, if the record number is 7 then the seventh record from the beginning of the file needs to be examined. If the record number is negarive then the it means to read the numbers relevant to the end of the file. For example, if the record number is -1 then the last record in the file needs to be examined. Thus, the command line task5 num.bin 1 2 -4 -8 34 means the program should use the file num.bin and it should examined the first, fourth and 34th record from the beginning of the file and the fourth and eighth record from the end of the file.