CP386: Assignment 2

This is a group (of two) assignment, and we will practice the concept of the parent-child process, inter-process communication, and some related system calls.

General Instructions:

- For this assignment, you must use C language syntax. Your code must compile using make **without errors.** You will be provided with a Makefile and instructions on using it.
- **Test your program thoroughly with the GCC compiler.**
- If your code does not compile, **then you will score zero.** Therefore, ensure you have removed all syntax errors from your code.
- Please note that the submitted code will be checked for plagiarism. By submitting the code file(s), you would confirm that you have not received unauthorized assistance in preparing the assignment. You also ensure that you are aware of course policies for submitted work.
- Marks will be deducted for any questions where these requirements are not met.
- Multiple attempts will be allowed, but only your last submission before the deadline will be graded. We reserve the right to take off points for not following directions.

Question 1

Create a C program ("collatz_sequence.c") that generates a collatz sequence for a given initial start number. The Collatz conjecture concerns what happens when we take any positive integer n and apply the following algorithm:

$$n=2n \quad , \text{if } \boldsymbol{n} \text{ is even}$$
$$n=3\times n+1 n=3\times n+1 \quad , \text{if } \boldsymbol{n} \text{ is odd}$$

The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1. For example, if n = 35, the sequence is **35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1**.

The program will use parent and child processes to create this sequence. The parent process will read a list of starting numbers from the file "start_numbers.txt" and store them in an array. For each number, the parent will create a collatz sequence and create a shared-memory object between the parent and child processes to pass the sequence to the child. The child will open the shared-memory object, output/print the sequence, and complete itself. Because the memory is shared, any changes the parent makes will also be reflected in the child process. The parent process will progress through the following steps for each number read from the "start_numbers.txt" file:

a. Establish the shared-memory object (**shm_open()**, **ftruncate()**, and **mmap()**).

b. Create a collatz sequence for the read number.

c. Save the sequence to a shared-memory object.

d. Create the child process and wait for it to terminate.

The parent will spawn multiple child processes (equal to the start numbers provided in the "start_numbers.txt") and pass different collatz sequence numbers to each child.

The child will progress through the following steps:

a. Establish the shared-memory object (**shm_open()**, **ftruncate()**, and **mmap()**).
b. Output the contents of shared memory.
c. Remove the shared-memory object.

The program exits when all sequences equal to start_numbers are printed on the console.

Use makefile to compile the program written above. The instructions to use and contents of the makefile have been provided on the MyLS course page. The other implementation details are at your discretion, and you are free to explore.

To invoke the program, use the command: ***./collatz_sequence start_numbers.tx***t in the terminal OR use the command: *make runq1* via *makefile*.

The expected output for Question 2:

```
$ make runq1
./collatz_sequence start_numbers.txt
Parent Process: The positive integer read from file is 10
Child Process: The generated collatz sequence is 10 5 16 8 4 2 1
Parent Process: The positive integer read from file is 53
Child Process: The generated collatz sequence is 53 160 80 40 20 10 5 16 8 4 2 1
Parent Process: The positive integer read from file is 99
Child Process: The generated collatz sequence is 99 298 149 448 224 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

Note: When submitting the source code files for Question 1, name them like:

- *collatz_sequence.c*

Question 2

In C, write a program (name it "process_management.c") that will involve using the UNIX fork(), exec(), wait(), dup2(), and pipe() system calls and includes code to accomplish the following tasks:

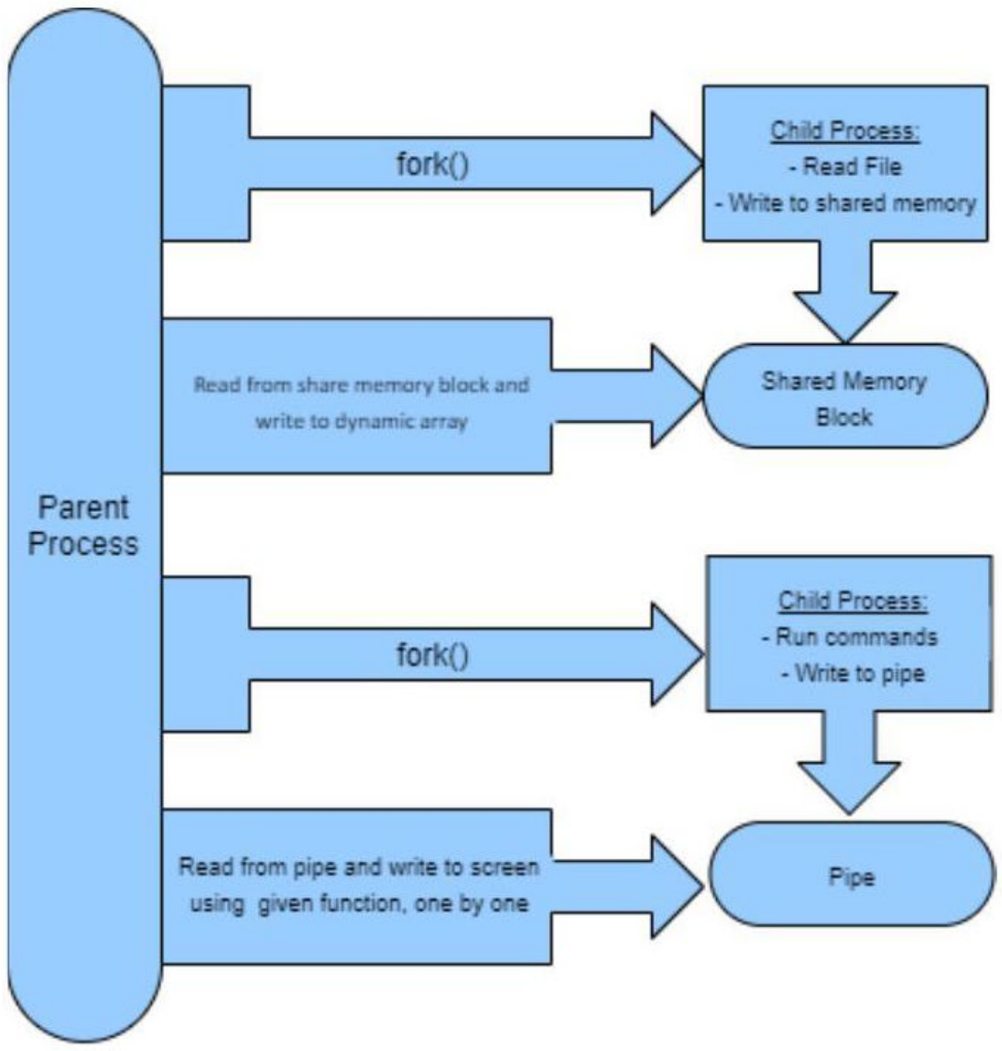
A parent process uses fork system calls for creating children processes, whenever required, and collecting the output of these. The following steps must be completed:

a) Creation of a child process to read the content (A LINUX COMMAND PER LINE) of the input file. The child process will retrieve the file contents in the form of a string using a shared memory area.
b) Creation of another child process(es) that will execute these Linux commands one by one. The process(es) will give the output using a pipe in the form of a string.
c) Write the output after executing the commands in a file named "output.txt" by the parent process.

The parent process can use a fork system call for creating children processes whenever required. The program flow is described below:

1. The parent process must use the command-line arguments to read the file name. The parent process would create a child process and it will read the contents of the file ("sample_in_process.txt"), (not the parent process).
    a) In this file, one shell command per line is present.
    b) The file's contents will be written to shared memory by the child process to allow the parent process to read it from there.
    c) After the file's contents are read, the child process will be terminated.
2. Then, the contents from the shared memory area will be copied to a dynamically allocated array and accessed by the Parent process.
3. Further, the following commands will be executed one by one during the child process:
    a) One or more child processes can be used by forking to execute the commands. For executing shell commands, **execvp()** system call can be used as **execvp(args[0], args)** and **dup2()** function by the child process, which duplicates an existing file descriptor to another file descriptor.
    b) For commands, e.g., **uname -a**, the output of the command to be written to a pipe by the child process, and the parent process will read it from the pipe and write it to a file via the output function. The given output function, **writeOutput()**, will be used by the parent process to write the output to a file ("output.txt"). The output function must be invoked iteratively for each command.

The flow of the program as follow:



Use makefile to compile the program written above. The instructions to use and contents of the makefile have been provided on the MyLS course page.

To invoke the program, use the command: *./process_management sample_in_process.txt* in the terminal OR use the command: *make runq2* via *makefile*.

Hint: The function for writing the results of executing the command you can use the function:

```
void writeOutput(char *command, char *output) {
    FILE *fp;
    fp = fopen("output.txt", "a");
    fprintf(fp, "The output of: %s : is\n", command);
    fprintf(fp, ">>>>>>>>>>>>>>>\n%s<<<<<<<<<<<<<<<\n", output);
    fclose(fp);
}
```

The program will create a file “output.txt”. The expected contents of output.txt are:

```
The output of: uname -a : is
>>>>>>>>>>>>>>>
Linux ubuntu 4.4.0-87-generic #110-Ubuntu SMP Tue Jul 18 12:55:35 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
<<<<<<<<<<<<<<<
The output of: free -m : is
>>>>>>>>>>>>>>>
              total        used        free      shared  buff/cache   available
Mem:            740         221          85           2         432         365
Swap:           765          18         747
<<<<<<<<<<<<<<<
The output of: ls -l -a -F : is
>>>>>>>>>>>>>>>
total 36
drwxrwxr-x  2 osc osc  4096 Feb 10 15:36 ./
drwxr-xr-x 12 osc osc  4096 Feb 10 15:35 ../
-rwxrwxr-x  1 osc osc 14088 Feb 10 15:36 a.out*
-rw-rw-r--  1 osc osc   431 Feb 10 15:36 output.txt
-rwxrwxr-x  1 osc osc  3057 Feb 10 15:34 process_management.c*
-rwxrwxr-x  1 osc osc    39 Feb 10 15:34 sample_in_process.txt*
<<<<<<<<<<<<<<<
The output of: whoami : is
>>>>>>>>>>>>>>>
osc
<<<<<<<<<<<<<<<
The output of: pwd : is
>>>>>>>>>>>>>>>
/home/osc/Assignment1
<<<<<<<<<<<<<<<
```

Note: When submitting the source code files for Question 2, name them like:

- *process_management.c*