CP386: Assignment 1 – Winter 2022

Due on January 29, 2022 (Before 11:59 PM)

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

General Instructions:

- Submit a zip file of your source code for each question and name it as combination of your Laurier ID number & your teammate's ID, an underscore, 'a' (for 'assignment'), then the assignment number in two digits. For example, if the users 100131001 and 100131233 are submitting Assignment 1 then it should be named as `100131001_100131233_a01.zip` that include each individual question's code.
- For this assignment, you must use C99 language syntax. Your code must compile using make **without errors.** You will be provided with a makefile and instructions on how to use it.
- **Test your program thoroughly with the GCC compiler (version 5.4.0) in a Linux environment.**
- If your code does not compile, **then you will score zero**. Therefore, make sure that you have removed all syntax errors from your code.
- Please note that the submitted code would be checked for plagiarism and by submitting this zip file you would confirm that you have not received any unauthorized assistance in the preparation of the assignment. You also confirm that you are aware of course policies for submitted work.
- Marks will be deducted from any questions where these requirements are not met.

Question 1

A zombie process is a process that has terminated but whose process control block has not been cleaned up from main memory because the parent process had not waited for the child. In C, create a program (named "z_creator.c") that forks a child process that ultimately becomes a zombie process. This zombie process must remain in system for at least X seconds (value of X you can choose.).

To terminate a Zombie process, we must first identify it using the command <ps -l>. The process states are shown below the S column; processes with a state of Z are zombies. The process identifier (pid) of the child process is listed in the PID column, and that of the parent is listed in the PPID column.

Write a second C program ("z_terminator.c") to automate the process to obtain the status of each process. Identify a zombie process by listing all running processes. Identify and kill its parent process. Print the updated list of all the other processes with their status.

Run the binary file of the above program (z_creator.c) to create a zombie in the background using command <./z_creator &>. We can identify the parent process ID of the zombie process by using command <ps -l| grep -w Z|tr -s ' '|cut -d ' ' -f 5>.

Once we have ID, we can use "kill -9 <parent process ID>" command to terminate the parent process. You can use the "system()" function for executing the Unix commands in a C program.

To invoke the program first use command: `./z_creator` and use `./z_terminator` in second terminal OR can use command: `make runq1` via `makefile`.

The expected output for Question 1

```
osc@ubuntu:~/a1$ ./z_terminator
In Parent Process!
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
In Child process!
4 S  1000  1147  1072  0  80   0 -  5656 wait   tty1     00:00:00 bash
0 S  1000  1360  1147  0  80   0 -  1054 wait   tty1     00:00:00 z_terminator
0 S  1000  1362     1  0  80   0 -  1088 hrtime tty1     00:00:00 z_creator
0 S  1000  1363  1360  0  80   0 -  1125 wait   tty1     00:00:00 sh
0 R  1000  1364  1363  0  80   0 -  7228 -      tty1     00:00:00 ps
1 Z  1000  1365  1362  0  80   0 -     0 exit   tty1     00:00:00 z_creator <defunct>

The updated list of processes and their status is:
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1000  1147  1072  0  80   0 -  5656 wait   tty1     00:00:00 bash
0 S  1000  1360  1147  0  80   0 -  1088 wait   tty1     00:00:00 z_terminator
0 S  1000  1372  1360  0  80   0 -  1125 wait   tty1     00:00:00 sh
0 R  1000  1373  1372  0  80   0 -  7228 -      tty1     00:00:00 ps
```

Question 2

In C, write a program (name it as “process_manangment.c”) that includes the code to accomplish the following tasks:

1. A parent process use fork system calls for creating children processes, whenever required, and collecting 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 file contents will be retrieved by the child process in the form of a string using a shared memory area.
    b) Creation of another child process which will execute these Linux commands one by one. The process 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.
2. The parent process can use a fork system call for creating children processes whenever required.
3. Chapter 3 in the book discusses POSIX API system calls for creating/using/clearing shared memory buffers.
4. The child process will read the file (“sample_in.txt”) contents, not the parent process. The command-line arguments must be used by the parent process to read the file name.
    a) In this file, one shell command per line is present.
    b) The file’s contents will be written to a shared memory by the child process to allow parent process to read it from there.
    c) After the file’s contents are read, the termination of the child process will be performed.
5. Then the contents from the shared memory area will be copied to a dynamically allocated array.
6. Further, the following commands will be executed one by the child process:
    a) For executing shell commands, `execvp` system call can be used.
    b) The output of the command to be written to a pipe by the child process and parent process will read it from pipe and write to a file via output function.
    c) One child process may be used by forking repeatedly to execute all the commands.
    d) 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.

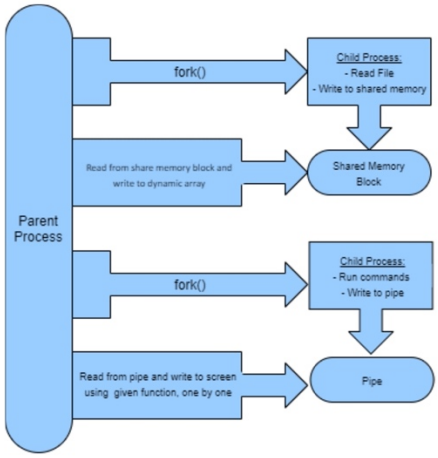

Figure 1: The flow chart

7. Figure 1 flow chart describes the flow of the program.
8. Use makefile to compile the program written above, the instructions to use and contents of the makefile have been provided in the Myls course page.
9. The other implementation details are on your discretion, and you are free to explore.

To invoke the program first use command: `./process_management sample_in.txt` in terminal OR can use command: `make runq2` via `makefile`.

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

```
void writeOuput(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 expected output for Question 2 is:

```
osc@ubuntu:~/a1$ ./processor_management sample_in.txt
osc@ubuntu:~/a1$ ls
Makefile  output.txt  processor_management  processor_management.c  q1  sample_in.txt  sampleName
osc@ubuntu:~/a1$ cat output.txt
The output of: free -h : is
>>>>>>>>>>>>>>>
              total        used        free      shared  buff/cache   available
Mem:           740M         46M        515M        2.5M        178M        572M
Swap:          765M          0B        765M
<<<<<<<<<<<<<<<
The output of: ls -l -a -F : is
>>>>>>>>>>>>>>>
total 60
drwxrwxr-x 3 osc osc  4096 Jun  1 15:14 ./
drwxr-xr-x 7 osc osc  4096 Jun  1 14:50 ../
-rw-rw-r-- 1 osc osc   270 May 23 18:48 Makefile
-rw-rw-r-- 1 osc osc   264 Jun  1 15:14 output.txt
-rwxrwxr-x 1 osc osc 17840 Jun  1 15:11 processor_management*
-rw-rw-r-- 1 osc osc  3057 May 15 12:43 processor_management.c
drwxrwxr-x 2 osc osc  4096 Jun  1 15:11 q1/
-rw-rw-r-- 1 osc osc    41 May 15 13:04 sample_in.txt
-rwxrwxr-x 1 osc osc  8760 Jun  1 14:53 sampleName*
<<<<<<<<<<<<<<<
The output of: whoami : is
>>>>>>>>>>>>>>>
osc
<<<<<<<<<<<<<<<
The output of: uptime -p : is
>>>>>>>>>>>>>>>
up 25 minutes
<<<<<<<<<<<<<<<
The output of: pwd : is
>>>>>>>>>>>>>>>
/home/osc/a1
<<<<<<<<<<<<<<<
osc@ubuntu:~/a1$
```