# CP386: Assignment 2 – Winter 2023

## Due on Feb 9, 2023 (Before 11:59 PM)

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.

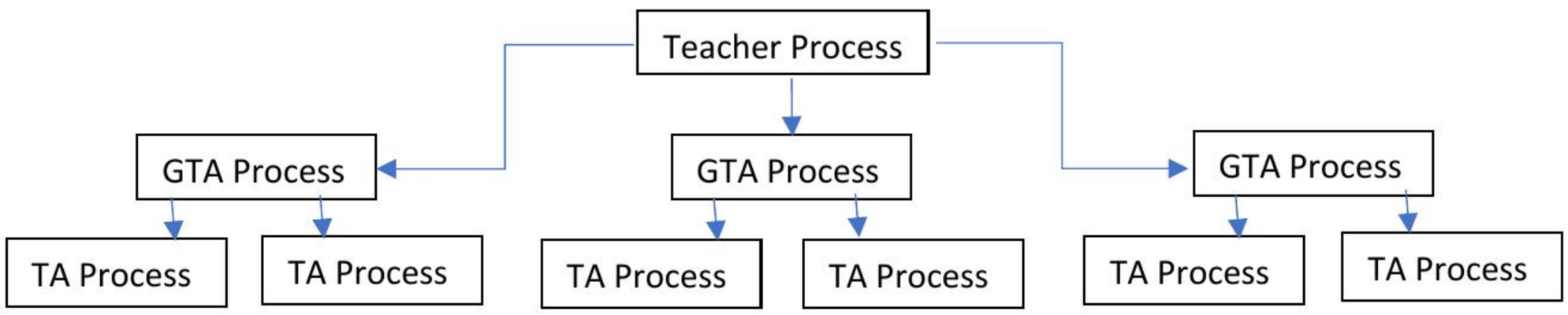
### General Instructions:

- 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 in a Linux environment.**
- If your code does not compile, **then you will score zero**. Therefore, ensure you have removed all syntax errors from your code.
- **Gradescope** platform would be used to upload the assignment file(s) for grading. The link to the Gradescope assignment is available on Myls course page. For submission, Drag and drop your code file(s) into Gradescope. **Make sure your file name is as suggested in the assignment; using a different name may score Zero.**
- 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 confirm 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 (name it "assignment_average.c") that finds the average grade of the course assignments. The teacher gives two assignments for every chapter. You must calculate the average grade for each assignment in all the chapters.

- The program should have a teacher process that reads the grades ("sample_in_grades.txt" grades file is available under the Assignment 1 section on Myls) of all assignments of all the chapters and creates a two-dimensional matrix of grades.
- Teacher process then creates a "GradTA" process. Each "GradTA" process takes care of solving a chapter.
- Each "GradTA" process would create "TA" processes and pass one assignment to each of them. "TA" process would calculate and print the average for that assignment. The process tree is shown below:




The input file "sample_in_grades.txt" contains data for three chapters and two assignments for each chapter (six columns). The first two columns are grades for two assignments for chapter 1, the following two columns are the grades for two assignments for chapter 2, and the last two columns are the grades for two assignments for chapter 3. The Grades file contains grades for 10 students. Make sure your program should dyanamically create GTAs and TAs processes based on the the number of the chpaters (you can assume that each chapter will always have two assingments).

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, first use the command: `./grades sample_in_grades.txt` in the terminal OR use the command: `make runq1` via `makefile`.

The expected output for Question 2:

```
Assignment 1 - Average = 10.200000
Assignment 2 - Average = 11.600000
Assignment 3 - Average = 13.700000
Assignment 4 - Average = 13.600000
Assignment 5 - Average = 9.400000
Assignment 6 - Average = 11.600000
```

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

- `assignment_average.c`

## Question 2

In C, write a program (name it "process_manangment.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 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(s) 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. Chapter 3 in the book discusses the POSIX API system calls for creating/using/clearing shared memory buffers. The program flow is described below:

1. The command-line arguments must be used by the parent process 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 termination of the child process will be performed.
2. Then the contents from the shared memory area will be copied to a dynamically allocated array and would be 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.

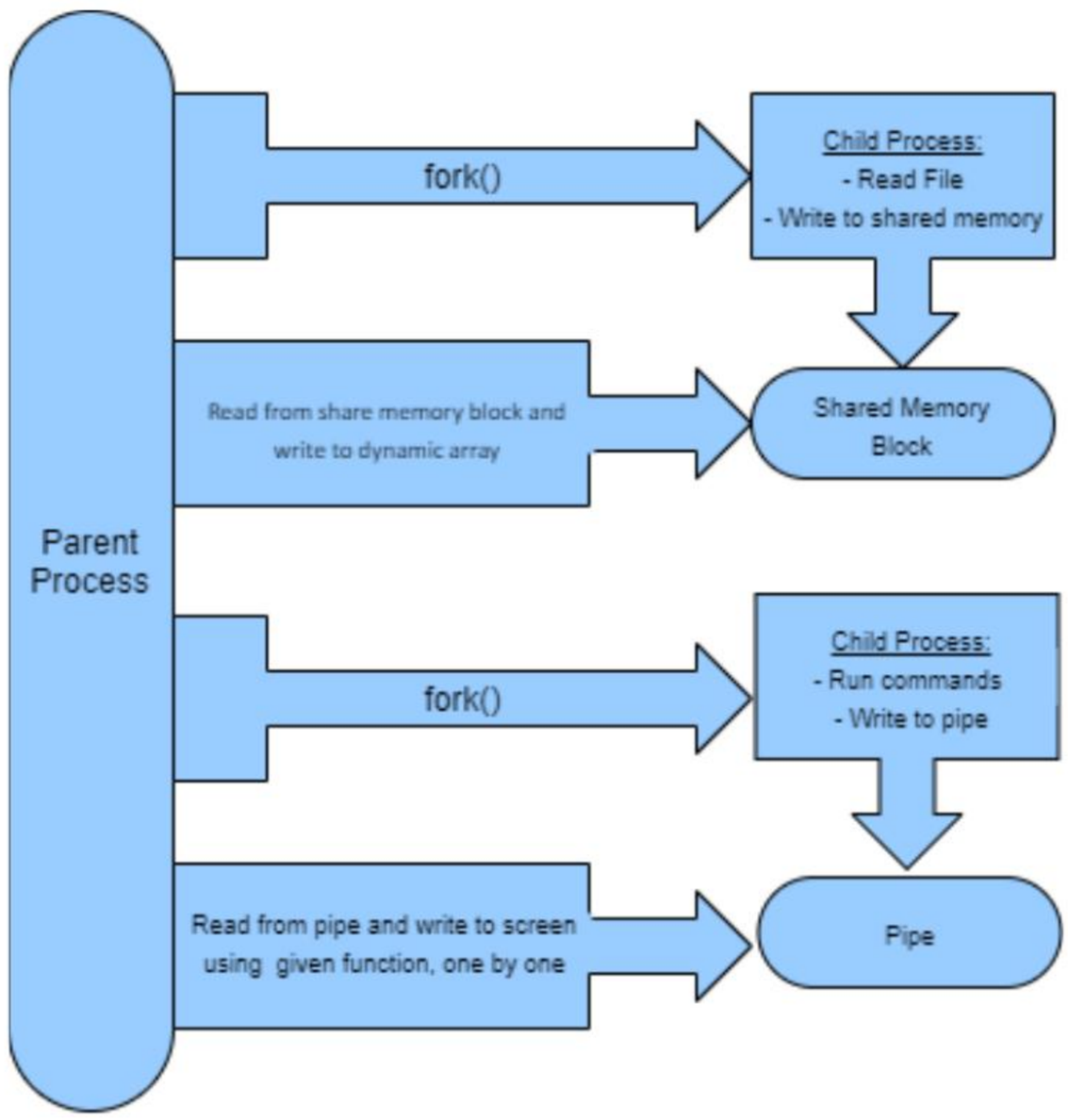



Figure 1: The flow chart

4. Figure 1 describes the flow of the program.
5. 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.
6. The other implementation details are at your discretion, and you are free to explore.

To invoke the program, first 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 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 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 source code files for Question 2, name them like:

- `process_management.c`