Programming Assignment #3

This assignment will give you some practice working with the POSIX shared memory API. Your task is to simply square an MxM matrix. For example, squaring matrix A will result in the following:

$$A = \begin{bmatrix} -4 & -1 & 2 & 1 \\ 6 & 0 & -9 & 10 \\ -3 & -10 & 6 & 0 \\ -10 & 6 & 7 & 9 \end{bmatrix} \qquad A^2 = \begin{bmatrix} -6 & -10 & 20 & -5 \\ -97 & 144 & 28 & 96 \\ -66 & -57 & 120 & -103 \\ -35 & -6 & 31 & 131 \end{bmatrix}$$

Another example:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad A^2 = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

The interesting part of this assignment is that each value in the resulting matrix will be calculated in a separate process. This means that if you have a 3x3 matrix, you will create 9 processes that will execute concurrently, one process for each of the 9 values. The parent process (the initial executable) will create the child processes. Each child process will *exec* a new executable program. The name of the child executable on the disk will be **`child-matrix`.** This program will perform the actual arithmetic of multiplying a row by a column. You will have a call similar to this in your child process:

```
char *args[6];           /* There are 5 args + NULL     */
args[0] = child_exe;     /* name of the program to exec */
        . . .            /* other args setup here       */
args[5] = NULL;          /* list terminator             */
execv(child_exe, args); /* overlay child with this     */
```

There are 5 parameters in the `args` array (plus a terminator) that will be passed to the **`child-matrix`**. (Examples using *execl* and *execv* were shown in class and are posted on the website for you to review. You can use either method.) The `child_exe` variable is demonstrated in the parent skeleton posted on the website. The parameters that will be passed to the program are:

Arguments (NUL-terminated strings)	Notes
0 – The name of the program on the disk.	This is the convention for `argv[0]`. This is supplied to the parent process on its command line. See `parent-matrix-skeleton.c`
1 – The id of the shared memory.	This was returned from **`shmget`** in the parent process. This is how the child will be able to access the shared memory setup by the parent.
2 – The child number.	This is a number from 0 to MxM – 1.This is so each child knows where in the matrix to put its result. See the diagrams on the web page.
3 – The row from the original matrix to work with.	This is an integer between 0 and M – 1. The child is simply multiplying one row by one column.
4 – The column from the original matrix to work with.	This is an integer between 0 and M – 1. The child is simply multiplying one row by one column.
5 – NULL	This will terminate the argument array.

Details
The parent must setup the shared memory for the interprocess communication between the parent and child. There are 3 "pieces" of information that will be written to and read from. The first 4 bytes of the memory are used to store the width of the array (integer). Since it's a square matrix, the width and height are both the same and we only need to specify one dimension. The next several bytes are the actual values (integers) of the matrix to square. Let's suppose that the matrix is a 3x3 matrix. Immediately following the first 4 bytes will be 36 bytes (`9 * sizeof(int)`), representing each of the 9 integers in the matrix. Row 1 will be written first, then row 2, and finally, row 3. Immediately following the elements of the input matrix are another 36 bytes which will be used by the child processes to save the results of their calculations. When all of the children have finished, the parent will then access the shared memory to read the values of the resulting matrix and print the matrix to the screen. See the website for more details.

Process Responsibilities
The parent and child processes have distinct roles.

The parent process is responsible for:

- Reading the input matrix from a file into it's non-shared memory.
- Printing the input matrix.
- Creating a shared memory area with **key of 123**.
- Writing the width of the matrix and all matrix values to the shared memory area.
- Forking all of the child processes, one for each element in the matrix.
- Waiting for all of the child processes to finish. The parent should be doing **nothing** while waiting for the children.
    - If the parent is doing busy waiting, you will lose points. The parent should just wait (block) until a child finishes.
- Printing the resulting squared matrix after all of the children have finished.
- Cleaning up any memory or resources that the parent needed, including the shared memory.

The child is responsible for:

- Determining which row number and column number to pass to the `child-matrix` program.
- Setting up all of the parameters to pass to the `child-matrix` program.
- *Exec*'ing (using *execl* or *execv)* the new child process (`child-matrix`).

Once the child process *exec*s the executable program, the child has now been replaced with a new process. This new process (`child-matrix`) is responsible for:

- Reading all of the command line parameters that were passed to it.
- Reading the width from the shared memory.
- Reading the correct row and column of the matrix from the shared memory.
- Multiplying the row and column together to get a single integer value.
- Writing the integer value to the proper location in the shared memory.
- Releasing any memory or resources needed, including the shared memory.

Notes
As in most systems programming, there is not a lot of code, but it can seem a little complex because most of you have never done this before. Finally, I expect all students to be using valgrind to check their programs for problems. This is an invaluable tool that can save you a lot of headaches tracking down memory leaks and other memory-related bugs. It can also track problems in child processes by using the `--trace-children` switch. Consult the man pages for details.

What to submit
You must submit your `parent-matrix.c`, `child-matrix.c`, `refman.pdf`, and the `typescript` file. These files must be zipped up and uploaded to appropriate submission page. Note that you are not submitting any other files.

Files	Description
parent-matrix.c	This is the source code to the process that will create multiple children. You must document the file (file header comments) and all functions (function header comments) using the appropriate Doxygen tags.
child-matrix.c	This is the code for the program that will be exec'd by the child processes. You must document the file (file header comments) and all functions (function header comments) using the appropriate Doxygen tags.
typescript	This is the session that was captured with the script command.
refman.pdf	The PDF produced from the Doxygen tags.

Make sure your name and other info is on all documents and files.