Assignment: Process Creation Due: 21 February

Processes in Linux are created using the `fork()` function to create a child process, and `exec*()` to replace the program code of the child with the code of a different program. In this assignment, you will familiarize yourself with process creation using `fork()` and `exec*()`, and basic process synchronization using `wait()`. These functions are described in the book and in the online Unix manual pages.

Specification

Your program is to take two matrices from an input file, multiply them and write the output to a file. Your implementation must use two programs, one that will execute as the parent process, and the other will be executed as the child processes to compute the dot product of a row and column vector.

Parent Program. The parent program must be named `matmult`, and must take three command line parameters:

```
matmult <infile> <outfile> <max-children>
```

where

- `<infile>` is the input file containing the matrices to be multiplied,
- `<outfile>` is the result of the matrix multiplication, and
- and `<max-children>`, is the maximum number of child processes that may be executing simultaneously.

For instance, the program could be run from the command line as

```
matmult p1in1.txt p1out1.txt 10
```

and your program would execute with no more than 10 child processes, reading from `p1in1.txt` and writing to `p1out1.txt`.

The `matmult` program should perform the following steps:

1. Determine the sizes of the matrices.
2. If the two matrices cannot be multiplied, print out an error and quit.
3. Create up to the maximum number of child processes, and assign to each the row from the first matrix, and the column from the second matrix to to be multiplied.
4. Wait for child processes to finish execution. Check their exit status to see if they terminated properly (see `man 2 wait`), and read the result of the vector product computed by the child process (more below). A message should be printed to standard output stating the child process id that completed, the row and column assigned to the child, and the resulting dot product.

5. If there are entries in the matrix multiplication left to compute, start a new child process to compute the next entry.
6. Return to step 4 until all matrix entries have been computed.
7. When all entries are computed, output the resulting matrix to `outfile`.

If any errors occur during the computation, print out a meaningful error message and stop multiplying the matrices. All executing child processes should be terminated using the `kill()` system call; and all temporary files used for communication between child processes and the parent should be removed.

Child Program. The child program `vecmult` must take three parameters:

```
vecmult <infile> <row> <column>
```

where

1. `<infile>` is the input file containing the matrices,
2. `<row>` is the row in the first matrix for multiplication,
3. and `<column>` is the column in the second matrix for multiplication.

The `vecmult` program should perform the following steps

1. Reads the specified row and column from the matrices in the input file.
2. After verifying that the input is valid, print out a message to standard output stating the process id, and the assigned row and column.
3. Compute the dot product.
4. Store the dot product in a file named `result-pid` for the parent process to read, where `pid` is the process ID of the child process. You can get the process ID with the `getpid()` system call.

When the child process terminates, the parent process should read the dot product result from the `result-pid` file and then remove the file after the result has been read.

If `vecmult` has any errors, such as incompatible vector size, running out of memory, invalid input file, etc., the program should print out a meaningful error message and terminate with a non-zero exit status. Use the `exit(...)` function to terminate with a non-zero exit status.

Input

The input file will consist of two matrices of the form

```
r c
a11 a12 ... a1c
...
ar1 ar2 ... arc
```

The first line contains the number of rows and the number of columns, and the remaining lines contain integer valued matrix entries. The second matrix will begin on the line immediately following the first.

Output

The output file should consist of one matrix in the form specified for the input file.

Submission

You are to submit a gzipped tar file of your solution to the curator at `http://spasm.cs.vt.edu:8080/curator01/`. In particular, your files should be contained in a directory named `pid-p1` where `pid` should be replaced by your university pid (my directory would be `bjkeller-p1`). The directory should contain

- A text README file that lists the files included in the directory, instructions for building the program, and instructions for running the program.
- The source code for your program(s).
- A shell script named `build` or a makefile for building your program.

Do not include files from an integrated development environment! Also do not include any object files or executables.

Submissions that are not in the specified form will not be accepted. It is your responsibility for confirming that your submission can be opened on the lab machines. Submissions that do not open will not be graded.

Demonstration

Your program will be graded by demonstration to a GTA on a computer in the 124 Unix lab running Mandrake 8.1 after first compiling with G++ version 3.0.3. Signup sheets for demonstrations will be passed around in class and then will be available in the Unix lab for late comers.

Minor fixes to problems will be allowed during a demonstration without penalty. Major fixes will be penalized by the late penalty in effect on the day of the demonstration. The GTA will have instructions on how to decide what is a major or minor change, and it will not be your call.

Implementation Requirements

Use of C++ I/O and string classes is permissible. Use of other template classes (STL or otherwise) is not.