CS3204 Operating Systems - Spring 2001
Instructor: Dr. Craig A. Struble
Project 1: Process Creation

Assigned: Wednesday, Jan. 24 **Due:** Last Demo Date: Feb. 9

1 Introduction

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.

2 Specification

In this assignment, you will be given two matrices to multiply. You are to write two programs

1. `matmult` which is the program for a parent process that executes child processes to multiply vectors in the matrices;
2. `vecmult` which is a program that computes the dot product of a row vector and a column vector for use in matrix multiplication.

2.1 Parent Process

The `matmult` program takes three parameters:

- `infile` is the input file containing the matrices to be multiplied,
- `outfile` is the result of the matrix multiplication
- and `maxchild`, is the maximum number of child processes simultaneously computing vector dot products.

The program is executed by calling

```
matmult infile outfile maxchild
```

replacing the parameters with appropriate values. 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, assigning a row in the first matrix and column in the second matrix to multiply together for each process.

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. 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`.

2.1.1 Error Handling

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 if an error occurs. All temporary files used for communication between child processes and the parent should be removed as well.

2.2 Child Process

The `vecmult` program takes three parameters:

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 program should be executed on the command line by

```
vecmult infile row column
```

The `vecmult` program performs the following steps

1. Reads the row and column to multiply from the input matrices.
2. Print out a message to standard output stating the process id of the program and the row and column assigned to multiply, after verifying that the input is valid.
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.

2.2.1 Error Handling

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.

2.3 I/O File Format

The matrices in the input and output file format are basically the same. Each matrix has the form

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

where the first line contains the size of the matrix in terms of rows and columns, and the remainder of the lines contain integer values for the entries of the matrix. The input file will contain two matrices and the output file will contain one matrix.

3 Submission

You need to schedule a demonstration time with GTA, where he will sit with you to test your program in the McBryde 116/118 undergraduate lab. A signup sheet for demonstration times will be made available in McBryde 133.

Sample files and execution command lines will be made available on the web site shortly. You should also generate your own test files and try your own combination of command line parameters to verify that your program works. For example, make sure your program works if the maximum number of children is greater than the total number of entries to compute in the matrix product.

We use the Curator, `http://ei.cs.vt.edu/~eags/Curator.html` to collect program submissions. The URL for submission is `http://spasm.cs.vt.edu:8080/curator/`. Only the servlet interface to the Curator is supported. Your project must be submitted before your scheduled demonstration time.

You are to submit a single tarred (`man tar`) and gzipped (`man gzip`) archive containing

- A text file named `README` describing the program, describing the contents of the archive, providing building instructions (including the platform you used for development), a user's guide (including how to start the program), and examples of usage with your test files;
- The source code for your program;
- The test files you used for your program;
- A script named `build` or a suitable `Makefile` for building your program.

Your files must be contained in a directory named `project0` of the archive. Be sure to include only the files listed above. Do not include extra files from an integrated development environment such as `configure` scripts, automake related files, etc. This is primarily an issue if you are using *KDevelop*.

Be sure to include your name in all files submitted. **DO NOT** include executables or object files of any type in the archive. **Be careful to FTP in binary mode if you are transferring your file to a Windows machine before submitting to the Curator. Invalid submission format will receive a grade of zero (0)!**

4 Programming Environment

As stated in the syllabus, you must use Linux and `gcc/g++` to implement this project. **All data structures used in your program must be student implemented. Using the standard template library (STL) or other third party libraries for data structure implementations is strictly prohibited.** Using C++ input and output streams and C++ strings is OK.