April 15th, 2020

CSE344 – System Programming - Homework #3 - v4
Fun with pipes and algebra :)

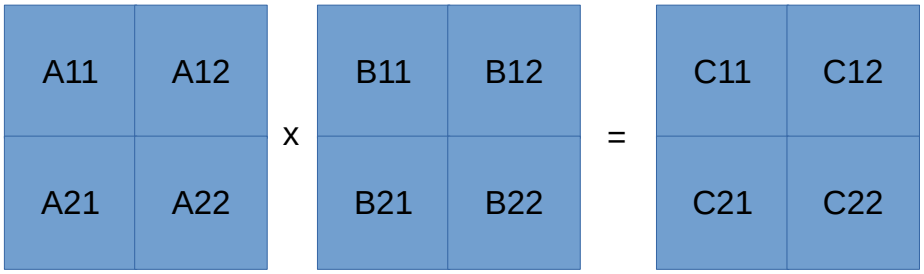
The program to be developed involves 5 processes: a parent P1 and its 4 children P2, P3, P4 and P5.

The main idea is for P1 to receive as input 2 square matrices, distribute the calculation of their product to its children, collect from them the partial outputs, combine them, and then calculate the singular values of the product matrix (you are free to copy paste from external sources for the singular value calculation stage).

The program (P1) will receive as command-line arguments the paths of 2 regular files and a positive integer n

```
./program -i inputPathA -j inputPathB -n 8
```

P1 will then read (2^n)x(2^n) characters from each of the input files. Each character it reads will be converted to its ASCII code integer equivalent. If there are not sufficient characters in the files then it will print an error message and exit gracefully. These numerical values will be interpreted as the rows of a (2^n)x(2^n) square matrix. Hence you will have read the contents of two matrices A and B; one from each input file respectively.



The goal now is to calculate C=AxB in a distributed fashion. There are many algorithms for parallel dense matrix calculation (look them up online if you are interested; parallelization is an exciting field with many opportunities); some are efficient, some...not so much. Since our objective is not parallelization but system programming we'll pick a simple and memory-wise inefficient one.

Assume the matrices are subdivided into quarters.

P1 will create 4 children processes and each of them will be responsible for calculating one quarter of C. P2 will calculate C11, P3 will calculate C12, P4 will calculate C21 and P5 will calculate C22.

In order to calculate Cij a process will need access to the i-th quarter row of A and j-th quarter column of B. In other words, if P2 is to calculate C11 of C, it will need access to the quarters A11, A12 and of course B11 and B21.

To achieve this, you will create **bi-directional** pipes between P1 and each of its children. A bi-directional pipe between P1 and P2, a bi-directional pipe between P1 and P3, and so on.

P1 will then send to each child process through the pipe the quarters of A and B that the child requires in order to calculate one quarter of C.

Then P1 must block its execution until all of its children have completed their calculations. This is a synchronization barrier. Use the pipe paradigm explained in our course to solve it.

Once all children have completed their calculations, P1 will collect their outputs through the pipes and form the product matrix C.

P1 must catch the SIGCHLD signal and perform a synchronous wait for each of its children.
In case of CTRL-C, all 5 processes must exit gracefully.

P1 will finallly calculate all the singular values of C, print them to the terminal and exit.

The children processes will wait for input from P1, process it, return the output to P1 and exit.

Rules:

- use the `getopt()` library method for parsing commandline arguments.
- Use system calls for all file I/O purposes
- Your programs are not allowed to crash due to any foreseeable reason. In case of an error, exit by printing to stderr a nicely formatted informative errno based message.
- If the command line arguments are missing/invalid your program must print usage information and exit.
- **Free all resources explicitly.**
- **No zombie processes!**
- All mathematical operations will be realized with an accuracy down to 3 decimal points.
- Do your best to produce a robust program that will not crash even with the most evil user executing it.
- Compilation must be warning-free.
- You will provide a demonstration of your program to the course assistants. They will provide the details.
- No late submissions will be allowed.

Submission:

- your source files, your makefile and a report on how you solved the issues in this homework.

Grading:

- Does not compile: fail
- Compiles but crashes despite normal input: fail
- Crashes for n reasons: -10 * n
- Violating the homework rules: fail
- No report: -30

Good luck.