Gebze Technical University Computer Engineering

CSE 344
System Programing

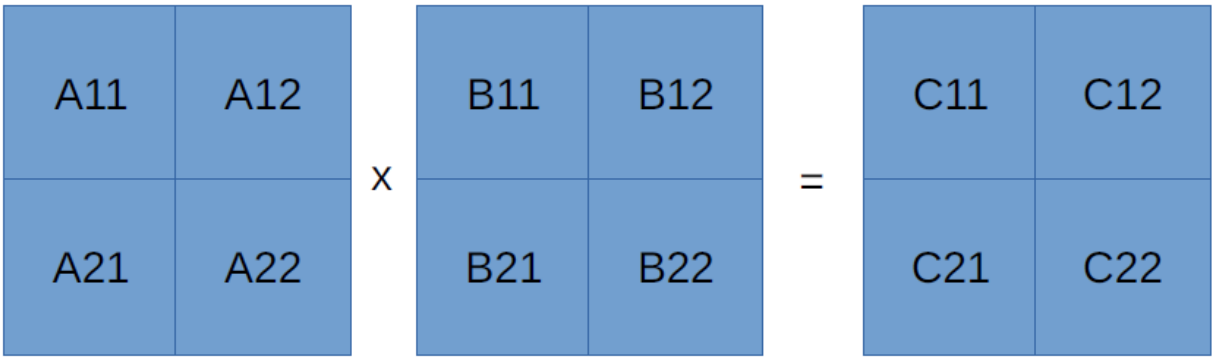
HOMEWORK 3
REPORT

ELİF AKGÜN
1801042251

In this homework, there are five processes: the parent process P1 and its child processes P2, P3, P4 and, P5. They run concurrently.

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 reads (2^n)*(2^n) characters from the input files denoted by inputPathA and inputPathB then it gets matrix A and matrix B. If there are not enough characters for given n, prints error message and exits. It converts the bytes to ASCII code.



The program multiplies the A and B matrices to get the C matrix. To do this, it divides the matrices to parts. Every process makes calculation of a quarter of C matrix. 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. Every child process takes necessary matrices elements from the parent then, parent takes parts of C matrix from the childs. Bidirectional pipes are used for IPC in this program.

First of all, I read input files denoted by inputPathA and inputPathB then I created two matrices: *inputA* (A matrix) and *inputB* (B matrix).

Then I created four matrices for every child processes (*rows_and_coluns_pn*). They include both matrix A's and matrix B's elements to calculate matrix C.

In this homework, for every child-parent communication I used 2 pipes so I used 8 pipes. For every child-parent relation, for bidirectional communication, there are a pipe child-to-parent and another pipe parent-to-child. I closed unwanted side of pipes.

In my program, I used for loop to create child processes. In every iteration one of childs is created and by default, parent process is created. Parent process writes the arrays denoted by *rows_and_columns_p2*, *rows_and_columns_p3*, and so on. Then it reads matrix C's parts from pipe. If there is no element to read, read system call wait until there is some elements in pipe or see end-of-file. This is how pipes works synchronously.
After reading pipes, parent creates matrix C from parts that calculated by child processes.

Child processes take matrices elements from pipe and calculate parts of matrix C with *calculate_c_matrix_elements* function. Every child process calculate one quarter of matrix C. After calculation, they write to pipe these parts with in array.

After child processes works are done, they send SIGCHLD to parent. By default, this signal ignores but in this program these signals are handled by *handler*. (SIGCHLD handler taken by course book.) To don't miss all signal, I used signal mask and I used sigsuspend to parent waits for children.
Last of the program, P1 finds singular values of product matrix and prints them on screen. To calculate singular values I used to svd function that copied from internet. Parent waits for childs to exit, then parent exits, too.

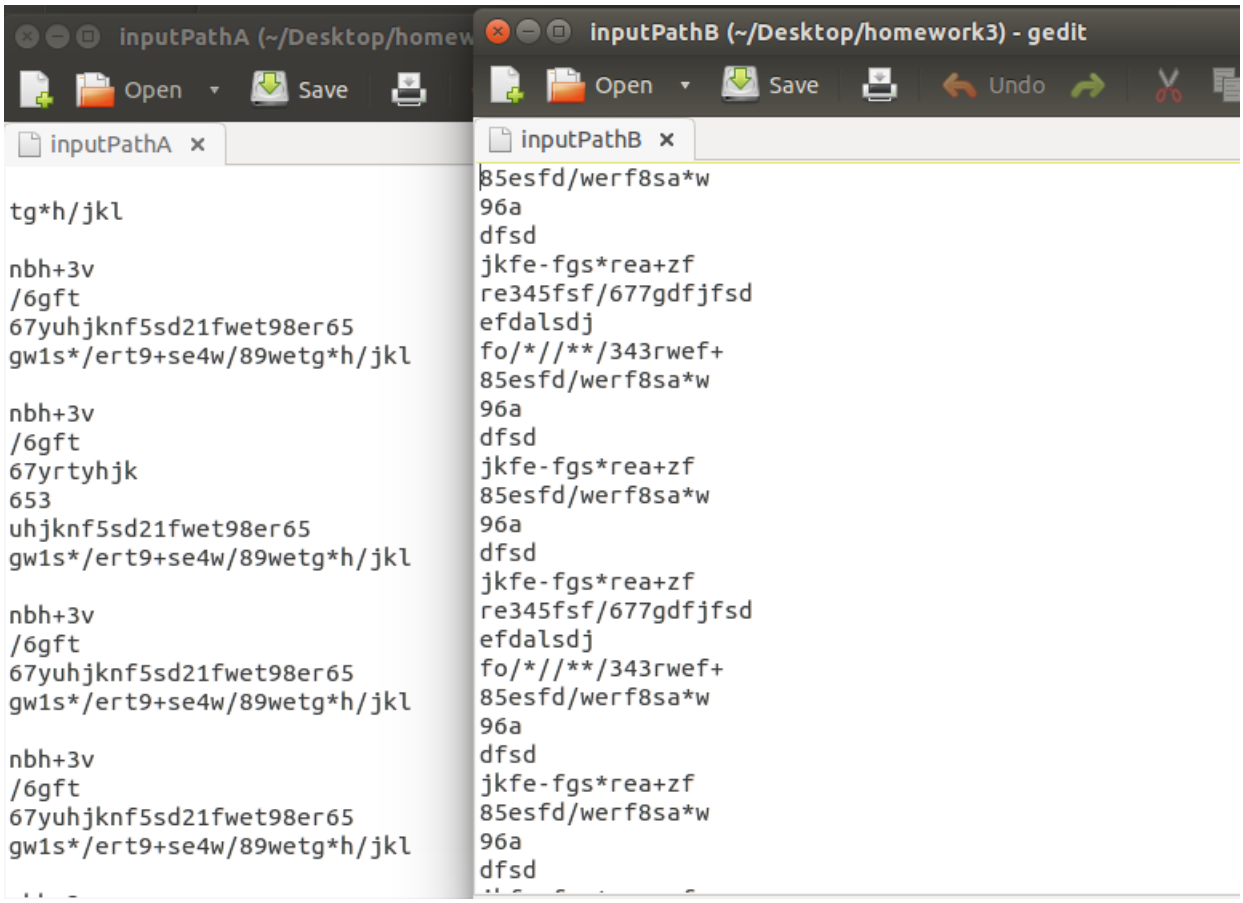
My program runs for n=1,2,…..,10. For n=11, I waited for minutes but I did not see anything. In case of ctrl+c, SIGINT caught and program exits.

Tests

1) In case of CTRL+C(SIGINT):

```
gtucpp@ubuntu:~/Desktop/homework3$ ./program -i inputPathA -j inputPathB -n 4
^CSIGINT is caught. Program exited.
gtucpp@ubuntu:~/Desktop/homework3$
```

2) Sample inputs and outputs



For n=1,2,3:

```
gtucpp@ubuntu:~/Desktop/homework3$ ./program -i inputPathA -j inputPathB -n 1
Singular values of matrix C:
534.879
23427.646

gtucpp@ubuntu:~/Desktop/homework3$ ./program -i inputPathA -j inputPathB -n 2
Singular values of matrix C:
106931.266
6616.198
528.854
2266.492

gtucpp@ubuntu:~/Desktop/homework3$ ./program -i inputPathA -j inputPathB -n 3
Singular values of matrix C:
439207.406
18317.855
10232.337
5861.797
4048.314
2623.944
381.075
1076.253
```

For n=4:

```
gtucpp@ubuntu:~/Desktop/homework3$ ./program -i inputPathA -j inputPathB -n 4
Singular values of matrix C:
1700529.750
42148.457
35246.000
29770.789
20985.965
19826.016
16358.758
13372.865
12243.553
8784.264
7252.388
5610.181
2854.091
34.597
858.895
2412.506
```

For n=5:

```
gtucpp@ubuntu:~/Desktop/homework3$ ./program -i inputPathA -j inputPathB -n 5
Singular values of matrix C:
6758386.500
89206.945
84000.438
78872.266
73190.789
63010.863
59046.832
54089.906
47198.070
42393.246
39787.035
34102.039
31298.322
27456.410
25401.348
22467.207
18302.061
12464.226
4867.419
11353.920
9465.786
9910.198
7104.360
0.002
0.002
0.002
0.001
0.001
0.001
0.000
0.000
0.000
```

3) Testing command line arguments:

As seen in the screen shot below if the command line arguments are missing/invalid my program prints usage information and exit.

```
gtucpp@ubuntu:~/Desktop/homework3$ ./program -i inputPathA -j inputPathB -n
Command line arguments must be this format to program runs correctly:
./program -i inputPathA -j inputPathB -n 8 (n must be positive)
gtucpp@ubuntu:~/Desktop/homework3$ ./program -i inputPathA -j inputPathB -n -4
Command line arguments must be this format to program runs correctly:
./program -i inputPathA -j inputPathB -n 8 (n must be positive)
gtucpp@ubuntu:~/Desktop/homework3$ ./program -i inputPathA -j inputPathB -n -- -7
Command line arguments must be this format to program runs correctly:
./program -i inputPathA -j inputPathB -n 8 (n must be positive)
gtucpp@ubuntu:~/Desktop/homework3$
```

4) Checking singular values:

I checked singular values from singular value calculator (*https://comnuan.com/cmnn01004/cmnn01004.php*), they are true.

There is an example for n=3.