Project 1

CS 433 Operating Systems

Assigned: Sat Oct 3, 2015

Note: This is a group project. All programs should be written in C or C++ language and compiled in a Linux distribution. You can work on this project in groups of one or two students. Each group needs to work on and code individually. No collaboration of any form is permitted and will amount to plagiarism if performed. Submit your .tar or .zip file, which contains all required files and a readme file, through the Cougar Courses.

Pipe programming

(**20 points**) Design a program using ordinary pipes in which one process sends a string message to a second process, and the second process reverses the case of each character and modify each character to the next one in the message and sends it back to the first process. For example, if the first process sends the message "Abcd", the second process will return "bCDE". Characters 'z' and 'Z' won't be changed. This will require using two pipes, one for sending the original message from the first to the second process, and the other for sending the modified message from the second back to the first process. Write this program using LINIX pipes.

Some other examples:
"Am Dx": the second process will return "bN eY".
"wxYZ": the second process will return "XYzz". (Characters 'z' and 'Z' won't be changed)
"Hello World": the second process will return "iFMMP xPSME".

Thread programming

(80 points) Matrix Multiplication Project

Given 2 matrices A and B, where matrix A contains M rows and K columns and matrix B contains K rows and N columns, the matrix product of A and B is matrix C, where C contains M rows and N columns. The entry in matrix C for row i, column j ($C_{i,j}$) is the sum of the products of the elements for row i in matrix A and column j in matrix B . That is,

$$C_{i,j} = \sum_{n=1}^{k} A_{i,n} * B_{n,j}$$

For example, if A is a 3-by-2 matrix and B is a 2-by-3 matrix, element $C_{3,1}$ is the sum of $A_{3,1} * B_{1,1}$ and $A_{3,2} * B_{2,1}$.

Matrices A and B can be initialized statically as shown below.

```
#define M 3
#define K  2
#define N  3

int A[M][K]={{1,4},{2,5},{3,6}};
int B[K][N]={{8,7,6},{5,4,3}};
int C[M][N];
```

In this project, your program should read the two matrices from files. Suppose we have 3*2 matrix A and 2*3 matrix B. The files have entries in the following format. The first line contains number of rows for matrix A and second line contains the number of columns for matrix A. It is then followed by data for matrix A with each row on a separate line. Similarly, the second file for matrix B. both files should be read as arguments when you run the program.

Here is how the two matrices files should look like:

3
2
1 4
2 5
3 6

2
3
8 7 6
5 4 3

In this project, you should read matrices A and B from files. Write a separate c program that creates two files for A and B containing random integers in range [1-100]. Then in your program, first read A and B from files and then create threads and calculate the product. Finally store the calculated C matrix in a third file.

For this project, write 4 versions of matrix multiplication program, run them and compare their performances.
1. Serial matrix multiplication with only one thread
2. Parallel matrix multiplication with M*N threads
3. Parallel matrix multiplication with T threads (divide C horizontally)
4. Parallel matrix multiplication with T threads (divide C vertically)

Run versions 3 and 4 in two different cases:
a. T is the core count in the system
b. T is twice the core count in the system

Parallel matrix multiplication with M*N threads

For this case, calculate each element $C_{i,j}$ in a separate worker thread. This will involve creating M*N worker threads. The main – or parent – thread will initialize matrices A and B and allocate sufficient memory for matrix C, which will hold the product of matrices A and B. These matrices should be declared global data so that each worker thread has access to A, B, and C.

The parent will create M*N worker threads, passing each worker the values of row i and column j that is used to calculate $C_{i,j}$. This requires passing two parameters to each thread. You can use, for example, the following **struct** to do that and pass this information to **pthread_create()**.

```
struct v
{
int i;       /*row*/
int j;       /*column*/
};
```

To create the worker threads, you can use a strategy similar to that shown below

```
/* creation of M*N worker threads */
for(i=0; i<M; i++)
  for(j=0; j<N; j++)
  {
    struct v *data = (struct v *)malloc(sizeof(struct v));
    data-> i = i ;
    data-> j = j ;
    /*  now create the thread passing it data as a parameter */
  }
```

You should figure out where is the right place for **pthread_join()**.

Parallel matrix multiplication with T threads (divide C horizontally)

Use C commands to get the number of cores available on your system. Then divide matrix C horizontally into T sections and then each thread calculates one section. For instance, if C contains 10 rows, and there are 2 cores in the system, the 1st thread calculates the 1st 5 rows and the 2nd thread calculates the last 5 rows.

Parallel matrix multiplication with T threads (divide C vertically)

Use C commands to get the number of cores available on your system. Then divide matrix C vertically into T sections and then each thread calculates one section. For instance, if C contains 10 columns, and there are 2 cores in the system, the 1st thread calculates the 1st 5 columns and the 2nd thread calculates the last 5 columns.

Notes:

1. Error messages should be displayed when an error occurs. For example, when the size of two matrices would make the multiplication impossible, such as $A_{4,5} * B_{2,3}$.

2. You can use your own method to pass parameters to each thread, the i and j.

3. Use time command to measure your program's performance (elapsed time).

4. Your program is graded based on its performance. Use time command to determine the duration of execution for each program. Compare the 6 given cases for each pair of A and B matrices given bellow. Plot an Excel chart to visualize your comparisons.

5. Check your program's performance for three sets of matrix productions:

```
1: A[2000*2000]  B[2000*2000]
2: A[2000*200]   B[200*2000]
3: A[1000*10000] B[10000*1000]
```

6. Submit a .zip or .tar file containing the following files:
   1. README.txt : Enter your names and explain how to run your programs.
   2. A Makefile to compile your programs.
   3. pipe.c
   4. create_matrix.c, serial.c, parallel.c, horizontal_parallel.c, vertical_parallel.c, performance.pdf