SJSU CS 149 HW2 SPRING 2021

REMINDER: Each homework is individual. "Every single byte must come from you." Cut&paste from others is not allowed. Keep your answer and source code to yourself only - never post or share them to any site in any way.

[Type your answer. Hand-written answer is not acceptable.]
[Replace YourName and L3SID with your name and last three digit of your student ID, respectively.]

1. (10 pts) Consider the following code segment:

```
    pid_t pid;
    pid = fork();
    if (pid == 0) { /* child process */
          fork();
          pthread_create( ... );
          fork();
    }
    pthread_create( ... );
    fork();
```

the remaining code (not shown) does not call `fork()` nor call `pthread_create()`. Assume each invocation to `fork()` or `pthread_create()` was successful.
a. How many unique processes are created (including the first main or root process)? Justify your answer.
b. How many unique threads are created by `pthread_create()`? Justify your answer.

Use a software to draw one single spacetime diagram (similar to the ones for fork2 and fork3 on slides, add a new notation of your choice to represent threads) to justify your answers for both a & b. Any software is fine. Hand-drawing receives no credit.

2. [programming question] (90 pts) You will write a multi-threaded application based on Pthread to perform parallel matrix multiplication. Q2 can be completed on Linux or Linux VM.

Matrix multiplication
To show how many rows and columns a matrix has we often write **rows × columns**. For example, the following is a 2 × 3 matrix (2 rows by 3 columns),

$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}$$

If **A** is an m **×** n matrix and **B** is an n **×** p matrix, the matrix product **C** = **AB** is defined to be the m **×** p matrix.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix} \qquad \mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

where
$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}$ for $i = 1, \ldots, m$ and $j = 1, \ldots, p$.

For more details, see the following links,
https://en.wikipedia.org/wiki/Matrix_multiplication
https://www.mathsisfun.com/algebra/matrix-multiplying.html

Partitioning the work
It is possible to compute each c_{ij} with one separate pthread. Therefore we have total m**×**p pthreads, all of which are running in parallel. It is also possible to compute all c_{ij} sequentially. Between these two, we can partition the work among a few pthreads which run in parallel. The partition can be based on rows, columns, or subgrids, and each

pthread simply computes one partition. To further simplify, we will adopt a row-based partition with the following rules:

i. Each row in the result matrix **C** is computed by the same pthread (except when the total number of thread is 0)
ii. Each pthread works on one or more rows in the resulting matrix **C**
iii. To better split the work evenly among pthreads, the difference between number of rows (in resulting matrix **C**) handled by any two threads should be 0 (when the number of rows can be evenly divided by the number of threads) or 1 (when the number of rows cannot be divided by the number of threads).

For example, if matrix **C** is 6 **×** 5 and given one thread, then this thread works on all elements of c_{ij} in sequential. For two threads, each thread works on 3 rows in **C**. For three threads, each thread works on 2 rows in **C**. For four threads, two threads work on 2 rows each, and the other two threads work on 1 row each.

As another example, if matrix **C** is 29 **×** 7 and given 5 threads, 4 threads work on 6 rows each, and the last thread works on 5 rows.

Matrix as 2-d Array in C

One may presentment the 4 **×** 3 matrix with a two-dimensional array in C as follows:

```
int matrix[4][3];
```

where the first index is the zero-based row number (from 0 up to 3) and the second index is the zero-based column number (from 0 up to 2). The following code initializes a two-dimensional 4 x 3 array:

```
int matrix[4][3] = {{1,2,3}, {4,5,6}, {7,8,9}, {10,11,12}};
```

where row 0 is `{1,2,3}`, row one is `{4,5,6}`, row two is `{7,8,9}`, and row three is `{10,11,12}`. In row 0, column 0, 1, and 2 have values 1, 2, and 3, respectively; In row one, column 0, 1, and 2 have values 4, 5, and 6, respectively; In row two, column 0, 1, and 2 have values 7, 8, and 9, respectively; In row three, column 0, 1, and 2 have values 10, 11, and 12, respectively.

Passing Parameters to Each Thread

The master will create all worker threads, passing each worker the configuration information, such as which rows should be computed. This step will require passing several parameters to each thread. The easiest approach is to create a data structure using a `struct`. For example, a structure to pass the row and column to a thread would appear as follows:

```
/* structure for passing data to threads */
typedef struct
{
   int row;
   int column;
} parameters;
```

With Pthreads, one can create worker threads using a strategy similar to that shown below:

```
parameters *data = (parameters *) malloc(sizeof(parameters));
data->row = 1;
data->column = 1;
/* Now create the thread passing it data as a parameter */
```

The data pointer will be passed to the `pthread_create()`, which in turn will pass it as a parameter to the function that is to run as a separate thread. You must deallocate each of the memory blocks by using `free()` after a thread completes.

In addition to configuration parameters in the `struct`, one can include additional parameters in the `struct` such as pointers to input and/or output matrices to each worker thread. If input and output matrices are global variables, each worker thread can reference input and output matrices directly without using pointers.

Input and Output

To compute the matrix **C** = **AB** where the matrix **C** is m **×** p, the user provides one command-line argument: the total number of threads (N) with the following semantics:

i. N == 0: compute <u>each</u> c_{ij} with one <u>separate</u> thread (0 ≤ i ≤ m-1, 0 ≤ j ≤ p-1). There will be m * p parallel threads.
ii. N <= m: partition the work subject to the rules illustrated in the section **Partitioning the work**. There will be N parallel threads.
iii. N > m: error out

Cut and paste the following code to your source code

```
#define     INPM1_ROW        5
#define     INPM1_COL        6
#define     INPM2_ROW        INPM1_COL
#define     INPM2_COL        3
#define     OUTM_ROW         INPM1_ROW
#define     OUTM_COL         INPM2_COL

#define     THREADCNT_ONEPERCELL  0

int inpm1[INPM1_ROW][INPM1_COL] = {{1,2,3,4,5,6}, {7,8,9,10,11,12},
{13,14,15,16,17,18}, {19,20,21,22,23,24}, {25,26,27,28,29,30}};
int inpm2[INPM2_ROW][INPM2_COL] = {{6,5,4}, {3,2,1}, {5,3,1}, {2,4,6}, {4,2,6},
{3,1,5}};
int outm[OUTM_ROW][OUTM_COL];
```

You are not allowed to have any additional global variables.

In the above code, `inpm1` and `inpm2` represent matrix **A** and **B**, respectively, and `outm` represents the matrix **C** = **AB**. The constant `THREADCNT_ONEPERCELL` is used to determine if the input argument (number of threads) is 0.

Based on the number of threads (a command line argument), your master program partitions the work among these threads and then creates these parallel threads, each passed with its own configuration.

Ensure
i. The partition algorithm is "generic". You can**not** hardcode the algorithm with expected number of threads.
ii. The actual matrix multiplication code is "generic", regardless of the number of threads from the command line. This can be done by having the master passes in any necessary configuration information via the `struct` to each thread.

Each thread must print one single line to stdout. This line includes the zero-based thread id and thread-specific configuration (i.e., parameters), in the following format
```
    thread[i]: param1=…, param2=…, …
```
where 0 ≤ i ≤ (number of threads) - 1. The master then waits for the termination of all threads and then prints the resulting matrix (row by row) to stdout before it terminates.

Compile your program with "`gcc -o matrix_multiply matrix_multiply.c -pthread`". You can execute the program in a Linux terminal with "`./matrix_multiply N`" where `N` is the integer value representing number of threads.

Reminders

1. Each invocation the program always prints out "CS149 Spring 2021 parallel matrix multiplication from FirstName LastName" only once to stdout. Your screenshot must include this line.

2. Any API in a multithreaded application must be thread-safe and reentrant (e.g., on Linux `rand()` vs `rand_r()`). Invoking any API that is not thread-safe or not reentrant is subject to deduction.

3. You must utilize array for various data structure and utilize loop to remove repeating code. Any repetitive variable declaration and/or code are subject to deduction.

4. Best practice:
a. include necessary header files only.
b. remove any warning messages in compilation.
c. error handling: always checks the return code from any API.

What you need to do in the report
a. (20 pts) Run your program 4 times in a Linux terminal, each with different number of threads

```
./matrix_multiply 0
./matrix_multiply 1
./matrix_multiply 2
./matrix_multiply 4
```

Include screenshots of your program execution. It is fine to split them into several screenshots.

b. (5 pts) Include code snippet and describe your partition algorithm. Illustrate your algorithm with example "29 rows with 5 threads".

c. (65 pts) source code as separate file.

====

Submit the following files as individual files (do not zip them together):

- `CS149_HW2_YourName_L3SID` (.pdf, .doc, or .docx), which includes
    - Q1: answers and justifications
    - Q2: perform a, b, and c as specified.
        - screenshots of the program execution. Screenshots that are not readable, or screenshot without "CS149 Spring 2021 …" from the program, will receive 0 point for the entire homework.
- `matrix_multiply_YourName_L3SID.c` Ident your source code and include comments.

The ISA and/or instructor leave feedback to your homework as comments and/or annotated comment. To access annotated comment, click "view feedback" button. For details, see the following URL:
https://guides.instructure.com/m/4212/l/352349-how-do-i-view-annotation-feedback-comments-from-my-instructor-directly-in-my-assignment-submission