Programming Assignment #3

Due date: 26.05.23, 23:55

In this assignment, we expect you to implement combined hashing and bubble-sort algorithms using Pthreads. You must sort the given random numbers by the following rules.

First, you will use the following additional data structures in your code:

```
/*@brief This structure must be used as nodes for the linked list.
* @r_node: must point to the right node of the linked list.
* @value: must hold the given number.**/
typedef struct node {
 struct node* r_node;
 unsigned value;
}node;

/*@brief This is the main hash table pointer. You must use a function to create a
*       double-linked list to index the corresponding linked list with modular
*      arithmetic (taking care of modulus/remainder value) for the corresponding
*       function.
* @node: Double linked list pointer. The first index of the storage 1D link list
*        must be calculated with modular arithmetic. The second will hold the
*        numbers whose modulus result is the same.
* @nof_element: must hold the total amount of the stored numbers. **/
typedef struct hash_table{
 node** list;
 unsigned nof_element;
}hash_table;

/*@brief To pass multiple structures to a thread during call/creating, you will
*         need a specialized structure holding all the required sub-structures.
*         You can design this structure as you wish. **/
typedef struct parameterPass{
}parameterPass;
```

We expect the following code flow for this assignment.

First, you must read all the numbers in Table 1 from the given CSV file **sequentially**. We expect you to define this section's two functions as **countNumOfElements** and **readNumbers**.

```
/** @brief, you must count the amount of numbers in the given CSV file.
  * @return value must hold the amount of numbers.
  * @filename: holds the name of CSV filename **/
unsigned countNumOfElements(char* filename);
```

```
/*@brief You will allocate an array with the amount of num_elements.Then, you
 *        will read all the numbers from the given .csv file into this array
 * @return pointer must point to the array allocated dynamically and hold the
 *        numbers.
 * @filename: must hold the CSV filename.
 * @num_element: must hold the amount of number in the given .csv file. */
unsigned* readNumbers(char* filename, unsigned num_element);
```

After these functions, you should have an array pointing to just the numbers (you can ignore number ids) in the CSV file.

Table 1: Given CSV file structure holding the numbers.

number_id	number
0	243355
1	196631
2	189151
3	131928

Next, you need to initialize the hash table. This hash table will consist of a double link list. The hashing function will be based on the modular of **(thread * (thread + 1)) / 2**. Here, you must define the hash table to cover all modular results. For example, if you are working with three threads, there can be 6 (0, 1, 2, 3, 4, 5) different modular results in total.

```
/* @brief You must initialize the basis 2D linked list structure. The outlier
 *        dimension must include (numOfThread * (numOfThread + 1))/2 indices.
 *        Each dimension will hold a linked list to store each modular result
 *        for each number respectively.
 * @numOfThread: is the number of threads entered by the user.
 * @numOfElements: holds the amount of numbers within the given CSV file.
hash_table *initializeHashTable(unsigned numOfThread, unsigned numOfElements)
```

Note: Each 1D linked-list points to NULL initially.

Afterward, you will create the structure (parameterPass) you will pass to each thread and call the first function to be executed by threads. This function is the **insertElements** function. In this function, each thread will index a certain array region, calculate the modular values of the numbers within the corresponding region and add them to the relevant location of the hash table. You can use the Pthread mutex functions to avoid the possible race condition here. However, when using mutexes, make sure to use them only in the regions of the possible existence of race conditions. Unnecessary or incorrect usage will result in a reduction on your overall grade. At this stage, you should work with the number of threads that is specified by the user. The array area where each thread will run will be calculated as follows:

```
interval = nof_element / nofThreads + 1;
unsigned offset = interval * tid;
unsigned last_ele = offset + interval;
if (last_ele > nof_element) last_ele = nof_element;
```

where **nof_element** holds the total amount of numbers, **nofThreads** holds the total number of threads specified by the user, and **tid** is the thread id that you set to each thread starting from zero. Thanks to insertionFunction, you will have a hash table storing all the numbers within the nodes. Even if you classify numbers by hashing function, the numbers are still unsorted within each 1D linked list.

The second sorting task uses the **bubble sort** algorithm to sort numbers within 1D linked lists. Different from the first function threads use, you must create (thread * (thread + 1)) / 2 number of threads and map them to each 1D linked list. Each thread will be responsible for sorting a 1D Linked list. As a result of this function, you should have a sorted hash table.

You must use the following function skeletons during the assignment:

```
void* insertionFunction(void *parameters); // 1st function to be used by
                                           // numOfThreads

void swap(node *a, node *b);  // used by the bubbleSort algorithm

void bubbleSort(node *start); // takes the 1D Linked list, and sort it depending
                              // on the node's values.
// 2nd function to be used by (numOfThreads + (numOfThreads + 1)) / 2 threads
```

- You will be graded depending on the memory leakage situation of your code. Hence, Release the unused memory portions while finalizing your code.
- You must take two command line arguments, the first for the **filename** of the name of the CSV file and the second one for the **numOfThreads**.
- Descriptions, variable types, and names are written to help you during the implementation of the assignment. You must write your codes depending on the given convention and rules.

Submission Rules:

- Please read all the instructions and guidelines related to the assignment carefully.
- Before asking your questions, discuss them with your friends related to unclear portions of the assignment. However, TAs will still try to answer your questions.
- The codes which give compilation errors will be graded as zero. If you think some of your functions run correctly, you can take partial grades. Hence, please do not submit the codes with compilation errors.
- Include your names, numbers (at the top), and comments on your codes.
- Cheating or any collaboration is strictly forbidden and results in a zero grade.
- Late submissions are not acceptable.
- Uploading one code for each pair will be enough through Teams before the due date.
- The uploaded file should be a **zip file** and contains student numbers, i.e., **<id1>_<id2>.zip**.

A sample execution

- ./executable_name <numbers.csv> <numOfThreads>