INFO0012-2/3 : Second Project
(Parallel programming)

Deadline December 14th, 2018

General information

- **Deadline: December 14, 2018, 23:59**.
- Project must be done by **teams of two students**.
- There will be a penalty for late submission.
- Questions will no longer be answered 24 hours before the deadline.
- English is strongly encouraged.
- Contact: `S.Hiard@uliege.be`, Office I.112 (B28).

1 Introduction and terminology

For this project, you are asked to write a program that sorts in ascending order an array of positive numbers using the *Radix Sort* algorithm. We will use the following terminology:

- **Radix (or Base):** The number of unique digits used to represent a number.

2 Radix sort implementation

Several implementations of the radix sort are possible. We will use the following.

Given that the program already received, via its arguments, the value N of numbers to sort, the array $number[\,]$ where $number[i]$ is the ith number to sort, and *base* which is the base (or radix) on which to sort:

1. The number n_{max} that has the maximum value is determined.
2. From this, we compute the number of iterations (which is equal to the number of digits of n_{max}).
3. A temporary 2-dimensional array $tempArray[\,][\,]$ is created, with *base* lines and N columns.

4. Each iteration consists in sorting the array on a given digit position, starting with the least significant one and ending with the most significant one.

5. At each iteration:

   (a) We browse $number[\,]$. For each number:

      i. We extract its current digit value d.

      ii. This digit is used as index in the 2D array, in which the number is stored, i.e. if we consider $number[i]$, whose digit value is d, and $number[\,]$ contains k numbers, prior to $number[i]$ that also have d as digit value, then the value of $number[i]$ is placed in $tempArray[d][k]$.

   (b) We concatenate the 2D array and store the numbers back into their original array. For each number, if they have the same digit value, they appear in the same order than in the original array, and if $d_{number[i]} < d_{number[j]}$, then $number[i]$ appears before $number[j]$.

6. The array is sorted.

3 Parallel program implementation

You are asked to write in C a parallel program that acts as described above. A single-process program performing a radix sort is provided.

- The array of number to sort is stored in shared memory. You are allowed to store the temporary array in shared memory as well.

- There will be as many processes as the radix. They will be referred to as "workers". Each worker is associated with the value of a digit (d_i) and works in two steps.

  1. It browses the original array $number[\,]$ and places the numbers in the temporary array $tempArray[\,][\,]$. Workers will try to split the work to be done, that is, given N numbers to sort and $base$ workers, each worker is responsible for approximately $N/base$ numbers. If $base > N$, then some workers will be idle in this phase. Particular attention should be given in the synchronization so that each number is correctly sorted in the temporary array. Additional information might have to be stored in shared memory to accomplish this.

  2. When $number[\,]$ has been completely browsed, each worker provides to an additional "master process" the amount of numbers

that appear in the $(d_i + 1)^{th}$ line of the temporary array. It then receives, from this master process, the position where to start writing the numbers back into *number*[ ].

- All processes have access to the data stored in shared memory.
- Access to the shared memory will be controlled with semaphores, if needed. You are not allowed to use active waits (i.e., repeatedly testing a condition in a loop).
- Communication between the master process and the worker processes is handled using message queues only.

4 Submission procedure

- This project must be coded in C using the `System V IPCs` for the shared memory, the semaphores and the message queues. The display will be performed in the console output.
- You must write a report describing how you implemented your program, and in particular how the synchronization is performed. This can be done by representing the program using the simplified C syntax used in the problem-solving lessons. You may use functions to simplify the pseudo-code in your report in order to focus on the inter-process communication.
- The project has to be done in teams of 2 students and completed before December 14th at 23h59. The completed program and report (PDF only) will be included in a ZIP archive named `sXXXXXX_NAME1_sYYYYYY_NAME2.zip` where `sXXXXXX`, `NAME1`, `sYYYYYY`, and `NAME2`, are the student IDs and uppercase surnames of the team members.

  Submit your archive to the **Montefiore Submission Platform**[1], after having created an account if necessary. If you encounter any problem with the platform, let me know (S.Hiard@uliege.be). However problems that unexpectedly and mysteriously appear five minutes before the deadline will not be considered. **Do not send your work by E-mail; it will not be read**.

Good programming !

[1] http://submit.montefiore.ulg.ac.be/