Yusuf Fawzy Elnady

Managing the Election Process (Parallel Processing)

(Message-Passing-Interface)

Description:

This is a **parallel program** simulates the **election process** to predicate which **candidate** will win and in which **round**.

Finally, it is time to vote for a **new president** and you are excited about that. You know that the **final results** may take weeks to be announced, while you can't really wait to see the results. Somehow you managed to get the **preferences list** for every **voter**. **Each voter sorted ou**t all candidates starting by his **most preferred** candidate and ending with his **least preferred** one. The voter always votes for the candidate who comes first in his preferences list.

For example, if the preferences list for one voter is **[3, 2, 5, 1, 4]** then voter will give the **highest vote** for candidate **3** and the **lowest vote** for candidate **4.**

Rules for the election process:

1. There are **C candidates** and **V voters**.

2. The election process consists of up to **2 rounds**. All candidates compete in the first round. If a candidate receives **more than 50% of the votes**, **he wins**, **otherwise another round takes place**, in which only **the top 2** candidates compete for the presidency, the candidate who receives more votes than his opponent wins and becomes the new president.

3. The **voters' preferences** are **the same in both rounds** so if the preference list [1 2 3 4 5] in the first round and the second round become between candidate 1 and 2 so the preferences is the same [1 2].

For example: If the input is

3 5 // number of candidates & number of voters
1 2 3 // voter 1 preference list
1 2 3 // voter 2 preference list
2 1 3 // voter 3 preference list
2 3 1 // voter 4 preference list
3 2 1 // voter 5 preference list
Then the output will be 2 2 // candidate 2 wins in round 2
Explanation: Candidate [1] got 40%, [2] got 40%, [3] got 20%. So second round will take place between candidates 1 and 2 with same preferences
1 2 // voter 1 preference list
1 2 // voter 2 preference list
2 1 // voter 3 preference list
2 1 // voter 4 preference list
2 1 // voter 5 preference list
Candidate [1] got 40%, [2] got 60% so candidate 2 wins in round 2

Yusuf Fawzy Elnady

Some Notes:

- The code generates **a data file** that contains the voters' preferences.
- When running the program, the file is not be loaded by one process, but **every running process loads its part from** the file by **seeking into it**, also the program handles the issue of having number of voters **not divisible** by number of processes.
- This big project is built using **MPI - C**.

The Implemented Parallel Scenario:

First, **the Master Process** will get the number of **candidates** and number of **voters** as an input from the user, then it calculates the **portion** and **remainder.** To handle the **file seeking** in order to let each process has its own part only from the file, I've made a function called **getLineLength()** which calculates how many **characters** actually in each line (assuming all lines are of the same length), then seek each time by the amount of **(lineLength * portion)**. Also there's another function called **getFirstTwo()** which calculates the number of characters in the **first two lines (C,V).**

To initialize the file with a **random data**, the **next task** of the **master process** is to fill the file with the help of the function I made called **shuffle(),** then writes these **shuffled numbers** into **the file**.

In the **broadcast step** I **share (the portion**, **no. of voters**, **no. of candidates** and the **line length)** among the all processes. Each process will **seek** to the portion it will **read** and save the data into its "**localArr"** then starts to calc number of winners in each line then we **reduce** all into one array called "**globalWin**".

The Master will handle the **remainder portion**, then it will calculate who's the first and second winner and determine if there're any **final winner** or we have to make a **second round**, if so again we need to **broadcast** the two winners to the all processes, letting them calculate the votes again. Finally, we will reduce the counts into two global Variables and see who will be the next candidate.

Installation Steps:

To install MPI, follow the instructions in this video link to let MPI be installed on Visual Studio, or you could run it on a virtual machine using the help of the file in this repo called **installation.pdf**, follow precisely the commands till you end up with complication command **"gcc -o out Election-Process.c"** then write **"./out"** to run the program.