Akhilesh Maddali(amaddali)
Christos Mitropoulos(cm1012)
Shubhank Varshney(sv441)

PROJECT 1
INTER PROCESS COMMUNICATION

<u>Design decisions and results:</u>

Part A:
This is a single process implementation where one process is created and does all the work i.e find the min, max, and sum. For small lists of numbers, this is the fastest implementation as there is no overhead to create addition process to do the additional work.

Part B:
For this part, we have implemented the below design.

```
proc0(read data from file)
|
|
|------->proc1(calculate max)
|          |
|          |--------->proc2(calculate min)
|          |            |
|          |            |----------->proc3(calculate sum)
|          |            |               |
|          |            |<--------------|
|          |<-----------|               x
|<---------|            x
|          x
x
```

The parent spawns a child process and the child then spawns another child process and so on. Each child process performs a specific calculation.
For large and small lists this implementation is fairly efficient but it is worse that part A as there is overhead to create the child processes.

Part C:
For this part, the parent spawns multiple children to complete the tasks of calculating the max, min, and sum. Here the parent first splits the main array into equivalent sub arrays. As each child is created, the parents passes the sub array to the child to calculate the max, min, and sum of the sub array. After each child has completed their task, inter-process communication was used where the child would pass the information back to the parent where the parent would collate the information and find the overall sum, min, and max.
This design is better than part B for larger files but for smaller files the efficiency is the same as part B.

```
proc0(read the file and calculate final max,min,sum)
  |
  |
  |
  |-------------------->proc1(calculate min,max and sum of partial list)
  |                      |
  |----------------------|
  |                      x
  |
  |--------------------->proc2(calculate min,max and sum of partial list)
  |                      |
  |----------------------|
  |                      x
  |
  |---------------------->proc3(calculate min,max and sum of partial list)
  |                       |
  |-----------------------|
  |                       x
  |
  |----------------------->proc4(calculate min,max and sum of partial list)
  |                         |
  |-------------------------|
  |                         x
  |------------------------>proc5(calculate min,max and sum of partial list)
  |                          |
  |--------------------------|
  |                         | x
  x
```

Part D:

```
proc0(read file and calculate sum)
|
|-----------proc1(calculate final max,min)
|       |
|       |
|       |
|       |-------------------->proc2(calculate min,max and sum of partial list)
|       |                      |
|       |----------------------|
|       |            x
|       |
|       |--------------------->proc3(calculate min,max and sum of partial list)
|       |                      |
|       |----------------------|
|       |            x
|       |
|       |---------------------->proc4(calculate min,max and sum of partial list)
|       |                       |
|       |-----------------------|
|       |             x
|       |
|       |----------------------->proc5(calculate min,max and sum of partial list)
|       |                         |
|       |-------------------------|
|       |                         x
|       |------------------------>proc6(calculate min,max and sum of partial list)
|       |                          |
|       |--------------------------|
|       |                  x
|-------------x
|
x
```

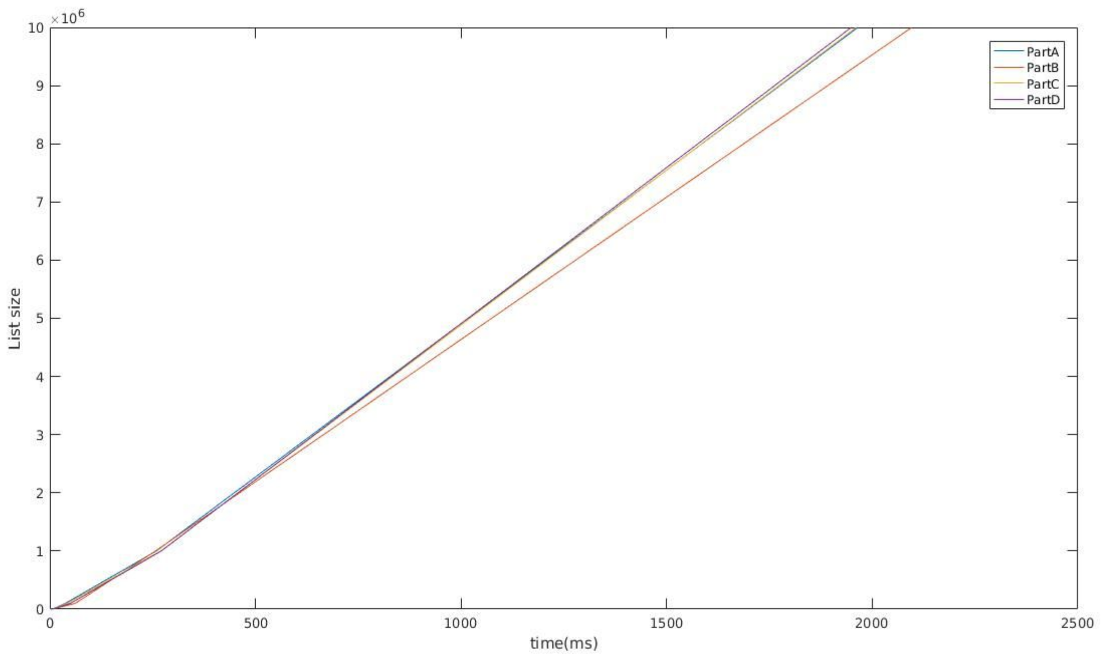
For this implementation, we have consolidated a few features in order to optimize the design. The first optimization we did was after reading the file into an array, we calculated the sum. What this does is, it reduces an entire for loop to calculate the sum which in turn reduces the total run time and reduces the overhead to create a new process to calculate the sum.
The second design implementation that we incorporated was inter-process communication (as done in Part C). The parent spawns a child process first. This child process then spawns a set required number of “grand-children” in order to perform the tasks of finding the max, min, and sum of the array. The child process splits the main input array into equivalent parts and passes each part to the grand child. The grand child then computes the max, min, and sum for the array segment assigned to them and passes that information back to the parent. The parent then collates the information from all the children and finds the sum, max, and min for the entire array. What this does is the work is divided equivalently among children which optimizes calculations rather than have one process do all the work. This is more of a divide-and-conquer technique. This design is not the best for the small lists of numbers but when there are large lists, the difference in processing can be seen (as evident in the graph provided below).

Overall Learning:
The results we obtained showed us that when the list size is small, having multiple processes and IPC to do the task results in larger amount of time as compared to the case when there is only a single process. This observation can be attributed to the fact that when the task is small, the time of completion is comparable to the overhead time of creation of multiple processes and communication between them. However, when the list size is large the benefits of multiple processes become evident and a better performance is achieved with a multiprocess IPC enabled program.

Extra Credit Information

	10	100	1000	10k	100k	1M	10M
PART A	2.5	3	4	10	38	259	1965
PART B	4	4	5	11.3	62	257	2096
PART C	4.2	3.5	4.2	11.3	42	270	1961
PART D	4	4	5	12	50	271	1949

Table contains the time taken (in ms) by each program for different list sizes. The results listed are an average of 3 readings.



Graphical representation of the data

We observe that the fastest implementation for List size of 10M is Part D, however PartA seems to be the best option when the list size gets smaller.