Operating Systems–1: CS3510
Autumn 2018

Programming Assignment 2 : Multi-Process & Multi-Threaded Computation of Statistics

Instructor : Dr. Sathya Peri
Author : Puneet Mangla (CS17BTECH11029)

---

1. **Design of Program :**

   From the input file given we created a array containing all observations. This array will be used finally for computations. The array need to be declared ***dynamically i.e in heaps*** because the array declared in stacks have size limit of 10^6 . So large input values cant be stored in stack.

   - **ProStat.cpp**

     For calculating the statistics using multi-processing , three child processes are forked from parent process. Each task is assigned to each child process. Since each process has its own copy of actual array there are no data conflict issues.

     Before forking the child process , parent process creates a shared memory instance that can hold three double variables. Each child process then puts its computed statistics in the shared memory in specific order.

     Parent process keeps waiting till all the child process are terminated . Once all are terminated , it resumes it works . Parent process links with shared memory and retreives the data . It opens a file and puts all data in it.

   - **ThStat.cpp**

     The variables holding mean , median and standard deviation are stored as global variables so that threads have access to them over their lifetime.

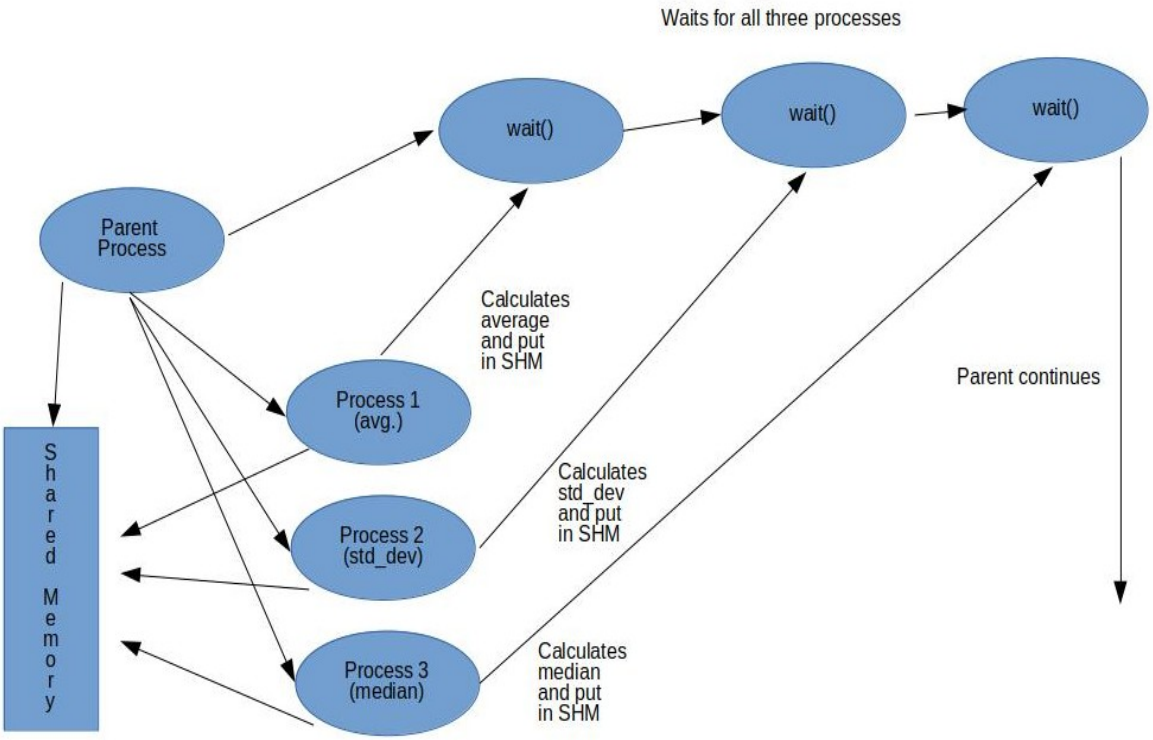
For calculating statistics using threads , parent defines three runner functions which will be assigned to individual threads that decide what they need to do when they are created. Three threads are created with default attributes and corresponding runner function is passed as an argument along with the array of observations.

For the thread which is calculating median we need to sort array . If we sort the same array then there will be consistency issues with other threads and output will get corrupted. To resolve this issue we made another copy of the array and used it to compute median.

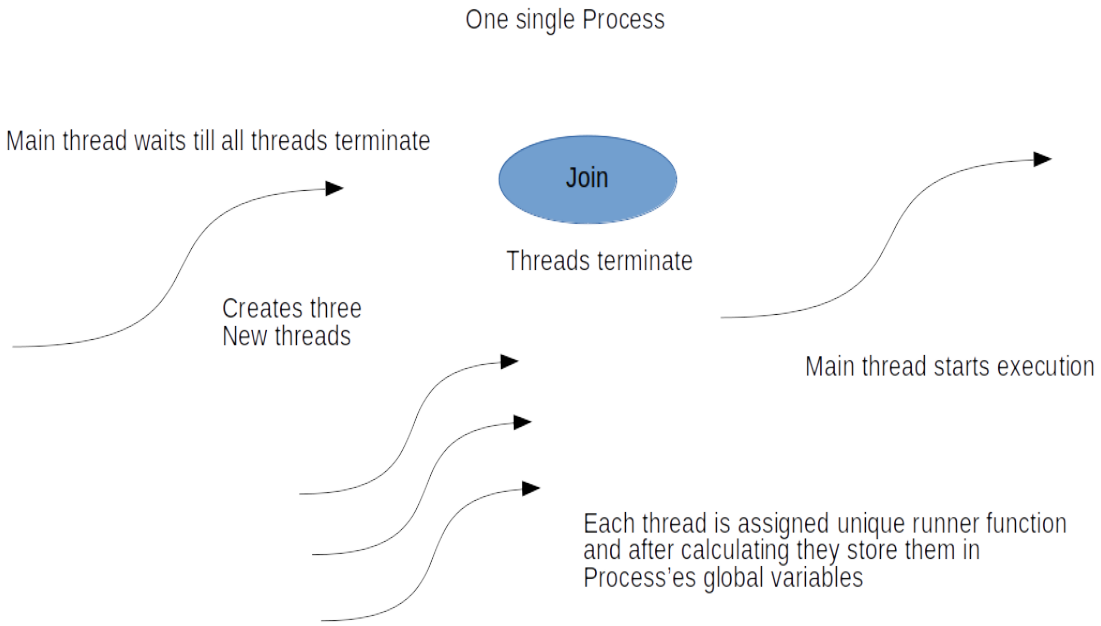
Main thread waits for all threads to finish and join it. After threads are terminated it creates a output file with required data.

The visual design of both programs is shown below :

- **Multi-Processing**

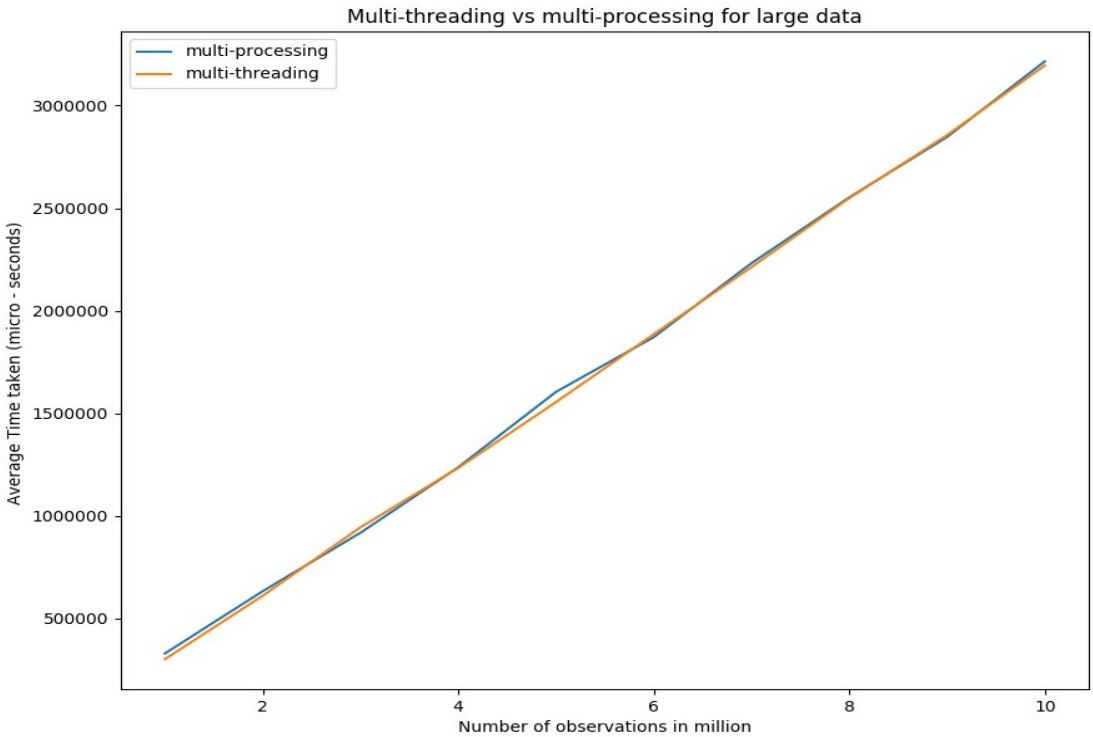


- **Multi-Threading**



2. Output Analysis :

Automated C++ script is written to generate a file with specified number of random numbers. The generated file is then used for analysis of time taken by both programs. The graph below summarizes it :



Form the graph it is clear that the time taken by individual program increases linearly with input data . This is because the computational complexity of finding mean and standard deviation is $O(n)$ and for median is $O(n\log n)$. Due to linear dependence on n , the graph is linear.

The time taken by multi-processing and multi-threading is almost similar , the reasons can be these :

1. The algorithm for calculating statistics in both program were same , taking almost same time in CPU.

2. The context switch overhead for processes maybe compensated by time taken to make another copy of array in thread.