Operating Systems–1
CS 3510 Autumn 2018
Programming Assignment 2:

Multi-Process & Multi-Threaded Computation of Statistics

INSTRUCTOR: DR. SATHYA PERI

Report By:
Vijay Tadikamalla
CS17BTECH11040

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Task

The goal of this assignment is to compute some statistics over a sequence of input numbers using multiple threads, processes and compare the times taken by each of them.

Approach

Multi-Process Program

1. Fork out three child process from the parent process.
2. Child processes are used to calculate mean, median and standard deviation
3. Once all the child process terminate, the parent will print the values of mean, median and standard deviation after reading from the shared memory.

Implementation

Header files

1. **#include<iostream>** it stands for Input Output Stream. It is basically a header file in c++ standard library.If you want to add streams in your program you need to include it. It provides basic input and output services for C++ programs
2. **#include <unistd.h>** defines miscellaneous symbolic constants and types, and declares miscellaneous functions.
3. **#include <fcntl.h>** defines the requests and arguments for use by the functions *fcntl*() and *open*().
4. **#include <sys/mman.h>** defines memory management declarations
5. **#include <sys/wait.h>** defines declarations for waiting
6. **#include <cmath>** It has mathematical function like pow()
7. **#include <algorithm>** It has algorithms like sort, nth_element()

To perform the above mentioned task, we develop a c++ program that is used to calculate mean, median and standard deviation.

We start our program by taking the input in the following way.

```
int n;
scanf("%d",&n);
double *arr=new double[n];
for (int i = 0; i < n; ++i)scanf("%lf",&arr[i]);
```

Input Details:

The input to your program will be a file containing the sequence of numbers. The first line of the file will be N which is the size of the input given. The second line will be list of input numbers. The following line shows an example:

5

45 3445 423 854 2

To achieve our goal, we need to establish a mode of communication between the child process and the parent process. So, we have to create a shared memory object with properties mentioned in the code snippet.

```
// The size (in bytes) of shared memory object
const int SIZE = 3*sizeof(double);;
// name of the shared memory object
const char *name = "/OS1";
// shared memory file descriptor
int fd;
// pointer to shared memory object
double *ptr;
```

shm_open creates and opens a new, or opens an existing, POSIX shared memory object. A POSIX shared memory object is in effect a handle which can be used by unrelated processes to mmap.

```
fd = shm_open(name,O_CREAT | O_RDWR,0666);
```

Syntax

```
int shm_open(const char *name, int oflag, mode_t mode);
```

ftruncate() configures the size of the shared memory object.

```
ftruncate(fd, SIZE);
```

```
ptr= (char *)mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
```

mmap creates a new mapping in the virtual address space of the calling process. The starting address for the new mapping is specified in addr. The length argument specifies the length of the mapping (which must be greater than 0).

Syntax

```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t
offset);
```

Now we create a Process Identifier variable pid. This helps us to identify the type of process by assigning each process a integer.

```
pid_t pid= fork();
```

- If pid==0, then it is a child process
- Else if pid>0, then it is a parent process

fork() creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process.

We can instruct the child process to execute the commands by adding a if condition of **(pid==0).**

So, using above mentioned approach we create three child process.

1. Child process with (pid==0) is used to calculate mean.
2. Child process with (pid1==0) is used to calculate median.
3. Child process with (pid2==0) is used to calculate standard deviation.

Calculating Mean

```
double Mean(double arr[],int n){
      double mean=0;
      for (int i = 0; i < n; ++i)mean=(mean/(i+1))*i+arr[i]/(i+1);
      return mean;
```

```
}
```

Time Complexity: O(n)
This implementation prevents the overflow.

Calculating Median

```
double Median(double arr[],int n){
      nth_element(arr,arr+n/2,arr+n);
      if(n&1)return arr[n/2];
      nth_element(arr,arr+n/2-1,arr+n);
      return (arr[n/2]+arr[n/2-1])/2;
}
```

Time Complexity: O(n)
Median can be calculated using many methods but the most efficient way to find median is tough to implement. So I have used the stl function nth_element.
std::nth_element() is an STL algorithm which rearranges the list in such a way such that the element at the nth position is the one which should be at that position if we sort the list.

It does not sort the list, just that all the elements, which precede the nth element are not greater than it, and all the elements which succeed it are not less than it.

It uses a mixture of deterministic quick select and median of median algorithm.

Calculating Standard Deviation

```
double std_dev(double a[], int n) {
    double mean=Mean(a,n);
    mean*=mean;
    double var=0;
    for(int i = 0; i < n; ++i)var+=(a[i]*a[i]-mean)/(n-1);
    return sqrt(var);
}
```

Time Complexity: O(n).
Here mean is calculated again because this child process runs parallel to the previous child process. So time taken to calculate the mean and time taken to read the the value of mean from the shared memory is almost the same.
sqrt() is a function of cmath header file.

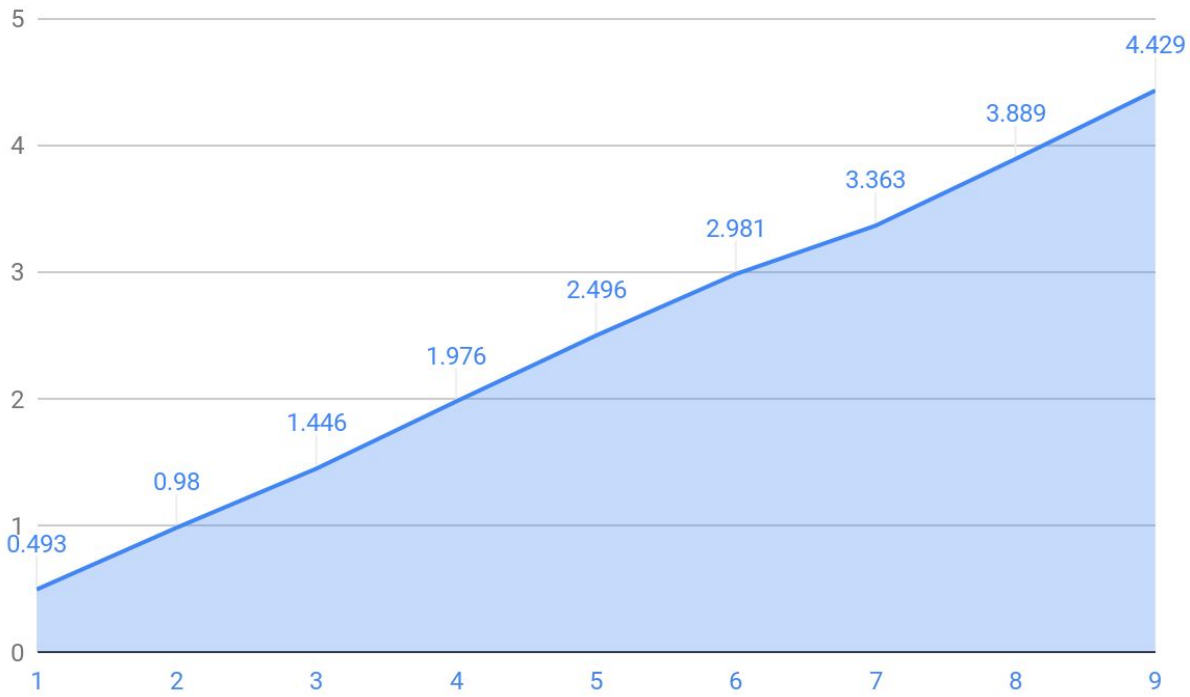
```
waitpid(pid,nullptr,0);
```

The parent process has to wait till all the child processes terminate. So the waitpid() is called. This function suspends the parent process till the child process gets completed. After all the child processes finish,parent process can print the value of mean, median and standard deviation by using the pointer to the shared memory object.

```
shm_unlink (name);
delete[] arr;
```

We can now remove the shared memory object and the memory allocated to the array by using the above command.

Graph(X-axis : Input Size(10^6), Y-axis: Time Taken(sec))



Design Decision

1. Size of memory object

2. Using double data type for preventing overflow.

Output analysis

1. Time increases linearly with the input size.

Program 2

Multi-Thread Program

Approach

1. Create three threads for calculating mean, median and standard deviation.
2. Join all the threads and report the value of mean, median and standard deviation.

Implementation

Header files

1. **#include<iostream>** it stands for Input Output Stream. It is basically a header file in c++ standard library.If you want to add streams in your program you need to include it. It provides basic input and output services for C++ programs
2. **#include <cmath>** It has mathematical function like pow()
3. **#include <algorithm>** It has algorithms like sort, nth_element()
4. **#include<pthread.h>** It has all the predefined functions for pthreads.
5. **#include <cstring>**This header file defines several functions to manipulate C strings and arrays. Ex Copying: memcpy: Copy block of memory (function)

To perform the above mentioned task, we develop a c++ program using pthreads and calculate mean, median and standard deviation.

We start by creating three threads and set it by using the default thread attributes.

```
pthread_t tid[3];
/* the thread identifier */
pthread_attr_t attr[3];
/* set of thread attributes */
```

We now assign task to each thread by passing it functions.

```
pthread_create(&tid[0],&attr[0],Mean,nullptr);
pthread_create(&tid[1],&attr[1],Median,nullptr);
pthread_create(&tid[2],&attr[2],std_dev,nullptr);
```

All the variables and arrays were declared global, so that each thread has access to all the Data.
The functions mean, median and standard deviations are the almost the same as described above.

```
void* Median(void *param){
	double *arr=new double[n];
	memcpy(arr,a,n*sizeof(double));
	nth_element(arr,arr+n/2,arr+n);
	if(n&1)median=arr[n/2];
	else{
		nth_element(arr,arr+n/2-1,arr+n);
		median=(arr[n/2]+arr[n/2-1])/2;
	}
	delete[] arr;
	pthread_exit (0);
}
```

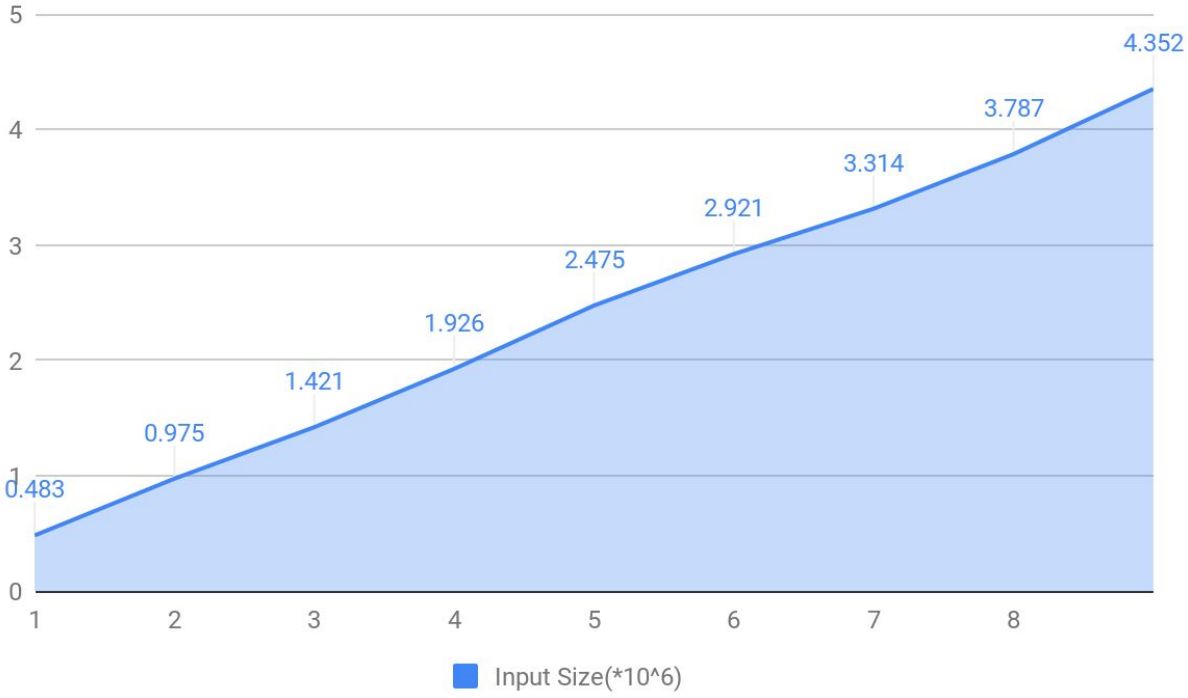
The only difference is that the median function creates the new copy of the array as it modifies it during it execution.

```
for (int i = 0; i < 3; ++i)pthread_join(tid[i],nullptr);
```

We have to wait for all the threads to get completed.

So, a simple method for waiting on several threads using the pthread_join() function is to enclose the operation within a simple for loop.

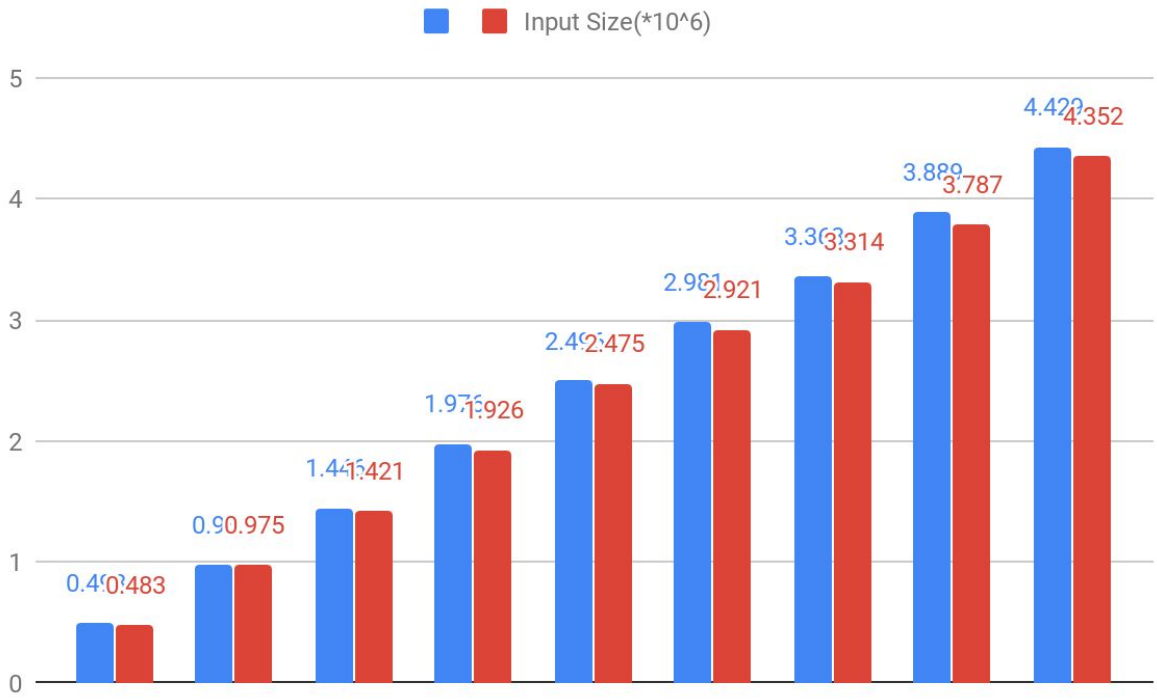
Graph(X-axis : Input Size(10^6), Y-axis: Time Taken(sec))



Output analysis

Time increases linearly with the input size.

Output Analysis



1. Time taken by threads is less than the process because process copy all the program blocks and data while threads share it.