Documentation

Assignment - 1

Operating Systems - 1

SAURAV VARA PRASAD CHANNURI
ES16BTECH11007

The question was to calculate mean, median and standard deviation using multiple threads and multiprocessing.

a) Multi Processing

In multiprogramming, we computed each function in each child process. We initially forked the parent process creating a child process. I then forked both the existing parent and child processes in-turn creating two more processes.

As a whole we have four processes, among which three are child processes and one is a parent process.

Each process has two pid values. For the parent process, the two pid values would be > 0

- Pid1 > 0 && pid2 > 0 - **parent process**
- Pid1 ==0 && pid2 > 0 - **child process 1**
- Pid1 > 0 && pid2 == 0 **- child process 2**
- Pid1 == 0 && pid2 == 0 - **child process 3**

We created shared memory by the following code

```
    int SIZE = 4096;
            /* name of the shared memory object */
     const char *name = "OS";

       int shm_fd;

       //struct timeval *start, end;

       long double *vals;

    shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

       ftruncate(shm_fd, SIZE);

       vals = (long double*)mmap(0, SIZE,PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
```

Where we initially create a size of 4 bytes, open a shared memory segment. Truncate the memory and point it using a pointer shm_fd of type int.

The shared memory is now allocated to the variable vals as we defined which is an array. We are going to store the output values of each function as each element of that shared memory array. We would have to create an entire Process control block which is exactly same as its parent process. This is an overhead of creation and this can be optimised by using multithreading.

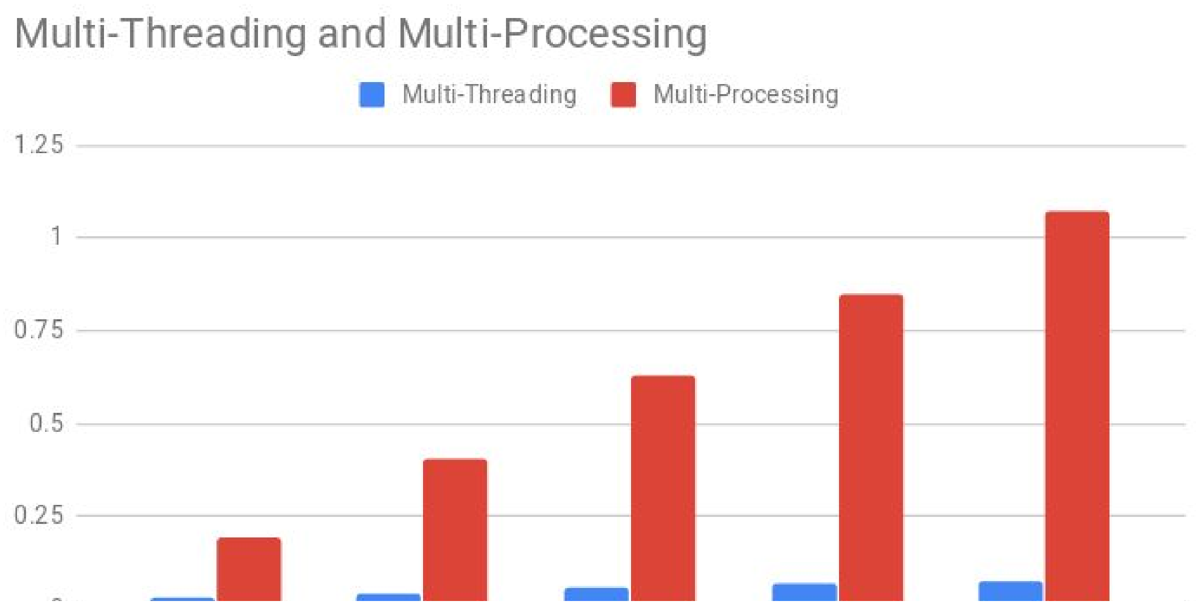
After all the three functions are executed, we printed all the results in the parent process. But the processes being different, we can't access the memory of another process. So, we create a shared memory and store the output values of each function in that shared memory.

We analyse the performance of computing these functions by noting the time taken to compute by each process, for arrays of different sizes from Size = **1*1000000 - 5*1000000**

b) **Multi-threading**

Now we compute the mean, median and standard deviation of a given array by creating three threads. One thread for each function. The outputs of these threads can be noted in a shared memory. Here the memory is shared using global variables as threads can share global variables as shared memory.

The time taken to create a process is larger than time takes to create a thread. We are operating on a single process so, we do not have to regenerate the entire **Process control Block** as in multi processing, but we just have to create **registers** and **stacks** for each thread, since the **code, data, files** will already have been shared among the following threads. Since the thread creation doesn't have this overhead of creation of



a new process control block exactly same as its parent process. **Thread creation is faster.**

The times taken for calculating all the three functions using multiprocessing and multithreading can be seen in the graph above.

In Multi-Processing, I applied task parallelisation and in Multi-Threading, I applied data paralellisation.

I used posix thread library to create multiple threads. First we have to define the pointers which point to a particular thread. The thread creation function **pthread_create(__ , __, __, __)** takes in four inputs . The first is the thread Identifier defined using **pthread_t.** The second input is for an attribute object. The third input is for the function we want to call which should be of **void* type** and the fourth input is the arguments of the function which should also be of the type **void*.**

After the creation of threads and the execution of the functions, we have to join the threads to a single process using the function
pthread_join(threadID)
Which waits until the given thread is finished to join the main process.

This way we can create multiple threads and execute our functions. And then give the output values stored in global variables at the end of the process.