OS Lab Assignment 2

Comparing runtimes for processes and threads for the same task (finding all primes below a number N using k threads/processes)

Using Processes:

We accept the user input *N* and *k*, and use them to spawn *k* processes -

```
        for(i = 0; i < k; i++){
                pid[i] = fork();
                if(pid[i] == 0) break;
        }
```

This way, each process that spawns has a value *i* that corresponds to its thread number, which we'll later use in calculating primes.
We also initialize a shared memory segment, into which we'll store the array of primes that each process finds.

```
        memd = shm_open("/shared_object", O_CREAT+O_RDWR, 0666);
        ftruncate(memd, k*N*sizeof(int));
```

We'll be writing a 2D array of size *k* x *N* to the shared memory, one row of length *N* for each process.

We use a switch-case block to determine which process is running -

```
switch(pid[i]) {
        case -1:
                // Return error

        case 0:
                // Child i

        Default:
                // parent
}
```

In the ith child, we check the numbers i, i+k, i+2k, i+3k…. Uptil N if they're prime. If they are, we store them in an array, which we write to the shared memory segment in the right location.

```
for(p = i+1; p < N; p += k) {
        for(p = i+1; p < N; p += k) {
                if(isPrime(p)) primes[r] = p;
                r++;
        }
}

memcpy(i*N*sizeof(int) + mmap(NULL, N*sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, memd,
0), primes, N*sizeof(int));
exit(0);
```

In the parent, we wait for all child processes to quit. After they quit, we read the 2D array, and print the numbers we find into a file PricPrimes.txt

```
int *n = mmap(NULL, k*N*sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED, memd, 0);
FILE *fp;
fp = fopen ("ProcPrimes.txt","w");
for(i = 0; i < k; i++) {
        for (j = 0; j < N; j++){
                if(*(n + i*N + j) != 0) fprintf(fp, "%d ", *(n + i*N + j));
        }
}
/* close the file*/
fprintf(fp, "\n");
fclose (fp);
```

We're done at this point. We close the shared memory object, and the parent quits.

```
        shm_unlink("/shared_object");
        printf("Contents written to ProcPrimes.txt, time taken - %f\n", t);
```

Using Threads:

We again accept *k* and *N,* and spawn *k* threads, each of which runs the *void *computePrimes()* function with it's thread number *i.*

```
        for (i = 0; i < k; i++) {
                pthread_create(&tid[i], NULL, computePrimes, (void*) i);
        }
```

We pass the thread number *i* as the argument to the function. We again use the same methodology of storing the primes and reading them - we use a 2D array of size *k*x*N* with each thread writing an array of primes of size *N* to the right location. Instead of the arrays being written to a shared memory segment, they're written to a global variable in this case.

```
void *computePrimes(void *x) {
        int i = (int) x;
        int k = params.k; int N = params.N;

        int p, q, *primes, r = 0;
        primes = (int *) malloc(N*sizeof(int));

        for(p = i+1; p < N; p += k) {
                if(isPrime(p)) primes[r] = p;
                r++;
        }
        memcpy(i*N + n, primes, N*sizeof(int));
}
```

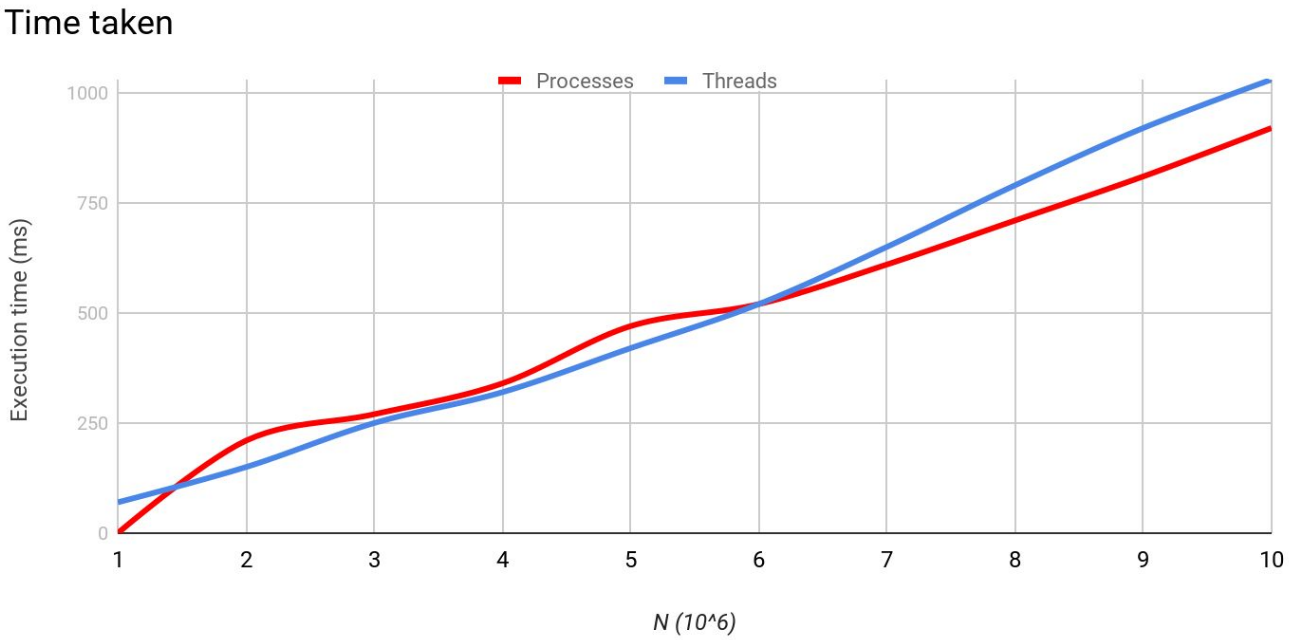
After execution, the threads are joined, and the 2D array is read and printed to the file.

```
        for (i = 0; i < k; i++) {
                pthread_join(tid[i], NULL);
        }

        FILE *fp;
        fp = fopen ("ThreadPrimes.txt","w");
        for(i = 0; i < k; i++) {
                for (j = 0; j < N; j++){
                        if(*(n + i*N + j) != 0) fprintf(fp, "%d ", *(n + i*N + j));
                }
        }
        /* close the file*/
        fprintf(fp, "\n");
        fclose (fp);
```

Comparison in Runtimes:

All runtimes were obtained by using the *time* call from the shell. Here are the runtimes -


Time taken
Processes
Threads
Execution time (ms)
N (10^6)