PLAGIARISM STATEMENT :
I certify that this assignment/report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment/report has not previously been submitted for assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, I understand my responsibility to report honour violations by other students if I become aware of it.

Name: JUPALLY SAI NISHITH
Date: 15-01-2020
Signature: JSN

Matrix Multiplication(Single Thread vs Multiple Threads vs Multiple Processes)

SINGLE THREAD:

In the single thread process i have allocated space for matrices A,B and C using malloc based on number of rows and columns obtained for A and B. Then i have initialized the matrix differently based on interaction mode status.

Then using 3 nested for loops i have done the multiplication part.

CODE SNIPPET:

```
for (int i = 0; i < arows; ++i)
  {
        for (int j = 0; j < bcols; ++j)
        {
                for (int k = 0; k < brows; ++k)
                {
                        temp = temp + (*(A+(i*acols+k)))*(*(B+(k*bcols+j)));
                }
```

```
                (*(C+(i*bcols+j))) = temp;
                temp = 0;
        }
  }
```

As per above code , at iteration i = x, j = y; We are calculating C[x][y] of product matrix. The variable k is used to traverse over x'th row of A and y'th column of B.Once the for loop of variable k ends, then temp is assigned to C[i][j] which is the required element of the product matrix.

By the end of all for all 3 loops multiplication is done.

For calculating time i have used gettimeofday() function. I placed a start variable before the for loops and an end variable after all the loops and calculated the time elapsed using these start and end variables.

I have made the memory used by A,B,C free.

MULTIPLE THREADS:

In the multiple thread process i have allocated space for matrices A,B and C using malloc based on number of rows and columns obtained for A and B. Then i have initialized the matrix differently based on interaction mode status.

I have created Z threads(i have assumed z to be number of threads/processes and i have declared Z as a macro for the flexibility of changing the number of threads).
I have created Z threads in a for loop and i have assigned the function called "Multithreadmul" to each of these threads.

CODE SNIPPET:

```
        for (int i = 0; i < z; ++i)
        {
                pthread_create(&threadmul[i],NULL,(void *)multithreadmul,NULL);
        }
```

The whole idea of multithreadmul function is that it gives "arows/z" number of consecutive rows to each a thread and z threads cover all the 'arows' number of rows. Obviously the scenario is different when arows<Z. But the whole idea of multi threaded process is make multiplication for large arrays faster because in case of smaller cases context switches time makes up a large chunk of total time and it is not useful.

So basically as per my function, each thread computes certain number of consecutive rows before it dies. I have used one semaphore mutex to avoid race conditions between threads when they are writing to the common variable threaditer(as per my code), which basically assigns each thread ,the rows to be computed by it. So i have used semaphore lock to avoid any errors.

CODE SNIPPET:

```
for (int i = x*arows/z; i<(x+1)*arows/z; ++i)
  {
        for (int j = 0; j < bcols; ++j)
        {
                for (int k = 0; k < brows; ++k)
                {
                        temp = temp + (*(A+(i*acols+k)))*(*(B+(k*bcols+j)));
                }

        (*(C+(i*bcols+j))) = temp;
        temp = 0;
  }
  }
```

In the above code, i = x*arows/z is starting row which is to be computed by a thread and i = (x+1)*arows/z is the last row to be computed by that thread. So we can see that it computes ‘arows/z’ number of consecutive threads and dies. The multiplication process is exactly the same as the one which is used in single threaded multiplication.

For calculating time i have used gettimeofday() function. I placed a start variable before the for loops(where thread is assigned a functions) and an end variable after all the loops in which all the threads join(obviously joining of threads happens once multiplication is done) and calculated the time elapsed using these start and end variables.

MULTIPLE PROCESSES:

In the multiple processes method, i had created three shared memory segments A,B,C of sizes required by A,B,C based on their number of rows and number of columns.

Then i have initialized the matrices differently based on interaction mode status.

Then for multiplication process, i have used the same logic of multiplication explained in single thread process and i have created the same number of processes as the number of threads.

I have created Z processes and for each process i have assigned some consecutive rows for multiplication.The loop gives “arows/z” number of consecutive rows to each a process and z processes cover all the ‘arows’ number of rows. In the for loop, at every iteration i have forked a new process and then for the child process i have assigned certain number of rows in the similar fashion to multi threaded process. This way every process gets certain number of rows to be computed and it dies once it is done computing the rows assigned to it.(i have used exit(0)).

Then in the parent process i have used wait(NULL), and waited till all the process are done. For computing the time i have used gettimeofday() function. I placed a start variable before the for loops(where new processes are forked) and an end variable after the for loop with wait(NULL)(obviously once the parent is done waiting multiplication is done) .

In each of the three functions i have returned (timeObtained)*(10^6) because the time expected is in micro seconds and time obtained by using gettimeofday() function is in seconds.

Exit_Failure:
In the main function before calling these 3 functions , i have checked if ‘acols’ is equal to ‘brows’ and if it is not, i have returned EXIT_FAILURE to the main.

OBSERVATIONS:

The speed up obtained in case of multiple threads is greater than the speed up obtained in the case of multiple processes as expected because threads are light processes and creating and switching between threads is much easier and faster when compared to creating and switching between processes.

As the matrix size increased, speed up obtained in case of multiple process and multiple threads increased with the usual trend of (speedUpofThreads > speedUpOfProcess) being common. But when matrix size is very small, the time taken in case of single threaded process was considerably small compared to other two routines. The creation and switching overhead in case of multiple processes and

threads for smaller matrices was dominating the computation overhead.So time for single threaded process was less compared to other two.

1. Interactive mode:

$ ./a.out --ar 3 --ac 3 --br 3 --bc 3 --interactive

SAMPLE INPUT and OUTPUT:

Enter A:

1 2 3

4 5 6

7 8 9

Enter B:

1 2 3

4 5 6

7 8 9

Result:

30 36 42

66 81 96

102 126 150

Enter A:

1 2 3

4 5 6

7 8 9

Enter B:

1 2 3

4 5 6

7 8 9

Result:

30 36 42

66 81 96

102 126 150

Enter A:

1 2 3

4 5 6

7 8 9

Enter B:

1 2 3

4 5 6

7 8 9

Result:

30 36 42

66 81 96

102 126 150

Time taken for single threaded: 2 us

Time taken for multi process: 1885 us

Time taken for multi threaded: 704 us

Speedup for multi process : 0.00 x

Speedup for multi threaded : 0.00 x

AS WE CAN SEE SPEED IS VERY POOR AS MATRIX SIZE IS SMALL

2. NON-Interactive mode:

To run:

$ ./a.out --ar 100 --ac 100 --br 100 --bc 100

SAMPLE INPUT and OUTPUT:

Time taken for single threaded: 4837 us

Time taken for multi process: 1088 us

Time taken for multi threaded: 828 us

Speedup for multi process : 4.45 x

Speedup for multi threaded : 5.84 x

AS WE CAN SEE SPEED IS GREAT AS MATRIX SIZE IS HIGH

These outputs reflect the above explanations given for small and big matrices.