Matrix Multiplication Using Single Thread,Multi Thread,Multi Process

1)In main check for the columns of matrix A and rows of matrix B if they are not equal exit(EXIT_FAILURE) otherwise continue the process.

2) Call the functions single_thread_mm(), multi_thread_mm(), multi_process_mm() which return the time for the process of calculating the matrix multiplication to the variables time_single, time_multi_thread, time_multi_process correspondingly and print the time taken to perform them and the speed ups correspondingly.

3)If interactive we ask the input from the user for each single thread,multi thread and multi process and print the corresponding output.In case of non interactive we call a function init_matrix for A and B matrix and perform matrix multiplication, the function called assigns a random value to the matrices A and B when called for each of them.

1)**Single thread:**

1)Allocate memory to the matrices based on the number of rows and columns of the corresponding matrix . For A = (int *)malloc(arows*acols*(sizeof(int))) similarly corresponding matrix sizes for B and C.

2)If interactive we take input from user for A and B matrix by calling the function input_matrix(A,arows,acols) and input_matrix(B,brows,bcols).If non interactive we call init_matrix(A,arows,acols) and init_matrix(B,brows,bcols) which initialises the matrices A and B to random values.

3)Initialise 'struct timeval' variables 'start' and 'end' which are inbuilt from library for calculating time. Start the time by using gettimeoftheday(&start,Null) and perform the matrix multiplication store the corresponding elements in the matrix C

```
for(i=0;i<arows;i++){

        for(j=0;j<bcols;j++) {

                *(C+(i*bcols+j)) = 0;

                for(k=0;k<brows;k++){

        *(C+(i*bcols+j)) = (*(C+(i*bcols+j))) +  (*(A+(i*brows+k)))*(*(B+(k*bcols+j))); }}}
```

4)Which performs the matrix multiplication and stores the values in C matrix. When multiplication is ended stop the clock by gettimeoftheday(&end,NULL). Use a variable t_sing to return the time for the function, that is the time taken for matrix multiplication of two matrices.We make some t_sing = (end.tv_sec - start.tv_sec)*1000000

t_sing = (t_sing + (end.tv_usec - start.tv_usec))

calculations in the code on the variable t_sing and get the time in micro seconds.gettimeoftheday( ) is used to get time and timezone.gettimeoftheday(struct timeval *start,struct timezone *tzone).Struct timeval gives the time in microseconds and seconds since its initialisation where we pass the arguments.tv_sec gives time and tv_usec gives in microseconds and the following calculation will return the time in microseconds when we return the whole function.

5)If it is interactive call the function output_matrix(C,arows,bcols). Freeing the memory assigned to matrices A,B,C and return the time t_sing.

2)Multi Threading:

1)Allocate memory matrices according to the sizes of the matrices A,B,C as satted similar in single thread function using malloc.

2)If interactive take the input for the matrices using input_matrix(A,arows,acols) and input_matrix(B,brows,bcols) if noninteractive the initialise the matrices using init_matrix(A,arows,acols) and init_matrix(B,brows,bcols).

3)Initialise nof_process to 11 which is constant to any size of the input which after analysing for various threads and various sizes observed faster to me.

4)Initialise an array to store the id of the threads created. Initialise the variables struct timeval start, end for calculating time and start the time gettimeoftheday(&start,NULL) and run a for loop with i as iterator for creating the threads pthread_create(&pid[i],NULL,(void *)mul,NULL) where void mul( ) is a function which is used for calculating the matrix multiplication and run a for loop for joining the threads pthread_join(pid[i],NULL) with i as iterator in the for loop and end the clock and calculate the time taken and store it in t_multiple_thread variable, the same calculations as above in single thread function.

5) If interactive print the output. Free the memory allocated to the matrices A,B,C and return the time value t_multi_thread.

6)The function void mul( ) is used for calculating the matrix multiplication and storing it in the resultant matrix and threads when returned we print the matrix.

7)Initialise a variable count, count = num1++ ,num1 is initialised to zero globally, this operation is performed as atomic operation using semaphores so that context switch does not happen in between while the count is increased as count needs to be implemented as an atomic operation, this count is used to give certain amount of the multiplication part to each thread and complete its work, like that all threads get some amount of work to do, some threads may not get any work to do and they exit, each thread perform their corresponding multiplication operation as follows:

```
sem_wait(&mutex); int count = num1++; sem_post(&mutex);

for(int i=count*arows/nof_threads;i<(count + 1)*arows/nof_threads;i++){

for(int j=0;j<bcols;j++){ *(C+(i*bcols+j)) ;

for(int k=0;k<brows;k++){

*(C+(i*bcols+j)) = *(C+(i*bcols+j))  +  (*(A+(i*brows+k)))*(*(B+(k*bcols+j)));}}}
```

As each thread gets its corresponding multiplication amount there won't be any problem when there is context switch in between and there are multiple threads and the corresponding result is returned into the matrix C correspondingly.

3)Multi Process:

1)In multi process we need to create shared memory so that it will be useful for the multiple processes to access and modify the memory and update in the matrices.

2)Initialise 3 variables shm_id1,shm_id2,shm_id3 which are used to store the id of the shared memory created by using shmget(IPC_PRIVATE, arows*acols*(sizeof(int)), IPC_CREAT | 0666) for shm_id1 which is created for matrix A and correspondingly for matrices B and C based on their sizes store their id's into shm_id2 and shm_id3 respectively. Allocate the memory created to the matrices to A,B,C by A=(int *)shmat(shm_id1,NULL,0), B = (int *)shmat(shm_id2,NULL,0), C=(int*)shmat(shm_id3,NULL,0) using these commands so that these shared memory variables can be accessed by the child processes and update them.

3)If interactive take input from the user by passing input_matrix function for matrix A and B correspondingly else initialise the matrices A and B values by the function init_matrix for each of the matrices. Make the nof_process as 5 constant number of process for any size of the matrix, similar to multiple threads case some of the process may or may not get any part of multiplication to complete.

4)Initialise the variable struct timeval start, end which are used to measure the time.Run a for loop as the nof_process with p as iterators before pid_t id =fork( ), use fflush(stdout) so that the cache memory does not get copied into the child process from the parent process and the parent memory output is printed out, creating child process as child has id value 0, write an if expression if(id ==0) then run a for loop for matrix multiplication which for a corresponding thread a particular part is given to complete the multiplication which is told by the outer for loop and the corresponding values by the child process are stored in the matrix C as follows:

```
if(id == 0){

for(int i=p*arows/nof_process;i<(p+1)*arows/nof_process;i++){

for(int j=0;j<bcols;j++){*(C+(i*bcols+j)) = 0;

for(int k=0;k<brows;k++){

*(C+(i*bcols+j)) = *(C+(i*bcols+j)) +  (*(A+(i*brows+k)))*(*(B+(k*bcols+j)));}}}exit(0);}
```

5)When a child is created the parent iterates over the for loop and create new child process, such that parent iterates and create child process and child performs the multiplication part given to them based on the for loop constraints such that anyone of the process does not intersect and overlap and the data is written into the matrix C.keep and exit(0) such that each process after completing its corresponding work it exits.

6)Outside the nof_process for loop, keep an another for loop such that it iterates for nof_process times and in that we keep wait(NULL) so that the parent process does not exit it waits for all the child process to exit. After the for loop gets completed we stop the clock and calculate the corresponding time as in single thread function above and store it in t_multi_process.If interactive then we print the matrix C by calling output_matrix(C,arows,bcols).

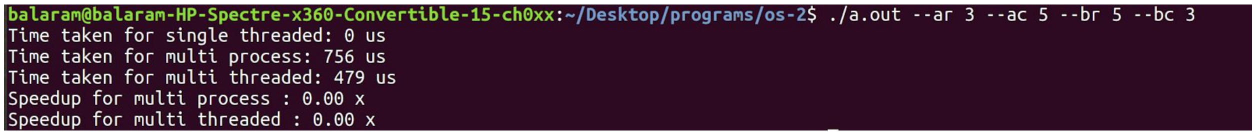
7)Detach the memory allocated to the matrices by using the command shmdt(A), shmdt(B), shmdt(C) and delete the shared memory created by shmctl(shm_id1, IPC_RMD, NULL), shmctl(shm_id2, IPC_RMD, NULL), shmctl(shm_id3, IPC_RMD, NULL) and return the time t_multi_process at the end.

Analysis of CPU

In my computer the file is compiled as gcc matmul.c

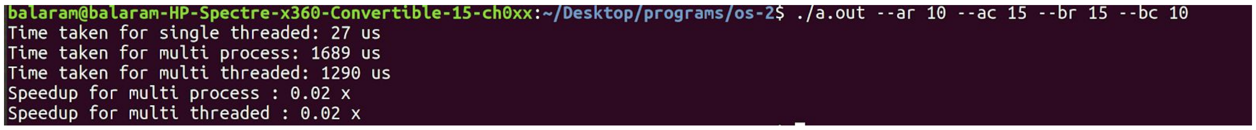
Run as: ./a.out `--ar <rows_in_A>  --ac <cols_in_A>  --br <rows_in_B>  --bc <cols_in_B> [--interactive]`

Test Case 1:



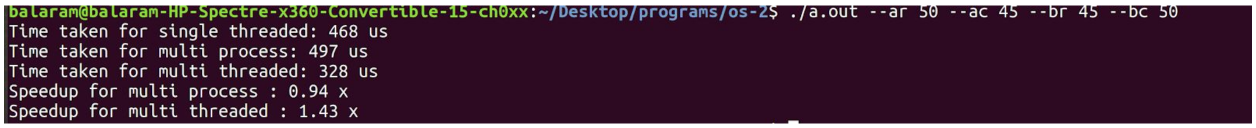
```
balaram@balaram-HP-Spectre-x360-Convertible-15-ch0xx:~/Desktop/programs/os-2$ ./a.out --ar 3 --ac 5 --br 5 --bc 3
Time taken for single threaded: 0 us
Time taken for multi process: 756 us
Time taken for multi threaded: 479 us
Speedup for multi process : 0.00 x
Speedup for multi threaded : 0.00 x
```

Test Case-2



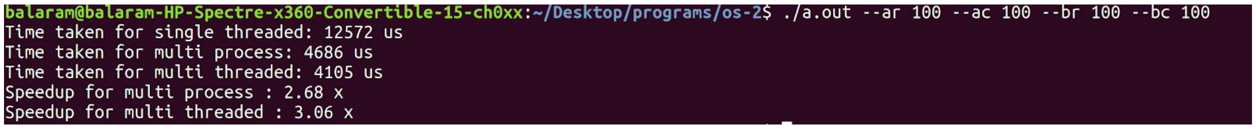
```
balaram@balaram-HP-Spectre-x360-Convertible-15-ch0xx:~/Desktop/programs/os-2$ ./a.out --ar 10 --ac 15 --br 15 --bc 10
Time taken for single threaded: 27 us
Time taken for multi process: 1689 us
Time taken for multi threaded: 1290 us
Speedup for multi process : 0.02 x
Speedup for multi threaded : 0.02 x
```

Test Case-3



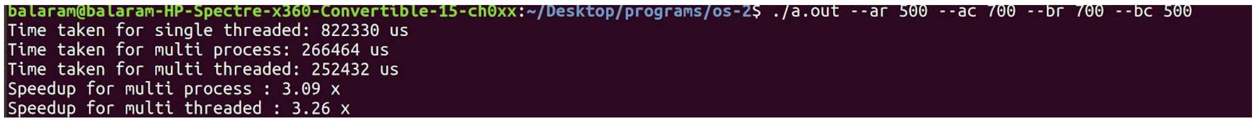
```
balaram@balaram-HP-Spectre-x360-Convertible-15-ch0xx:~/Desktop/programs/os-2$ ./a.out --ar 50 --ac 45 --br 45 --bc 50
Time taken for single threaded: 468 us
Time taken for multi process: 497 us
Time taken for multi threaded: 328 us
Speedup for multi process : 0.94 x
Speedup for multi threaded : 1.43 x
```

Test Case-4



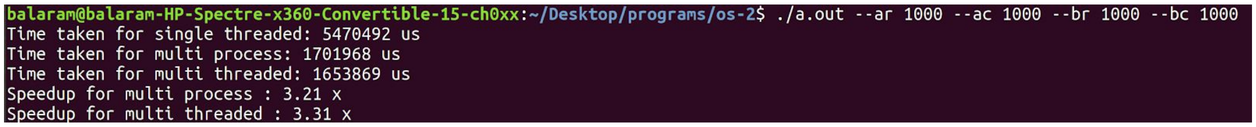
```
balaram@balaram-HP-Spectre-x360-Convertible-15-ch0xx:~/Desktop/programs/os-2$ ./a.out --ar 100 --ac 100 --br 100 --bc 100
Time taken for single threaded: 12572 us
Time taken for multi process: 4686 us
Time taken for multi threaded: 4105 us
Speedup for multi process : 2.68 x
Speedup for multi threaded : 3.06 x
```

Test Case-5



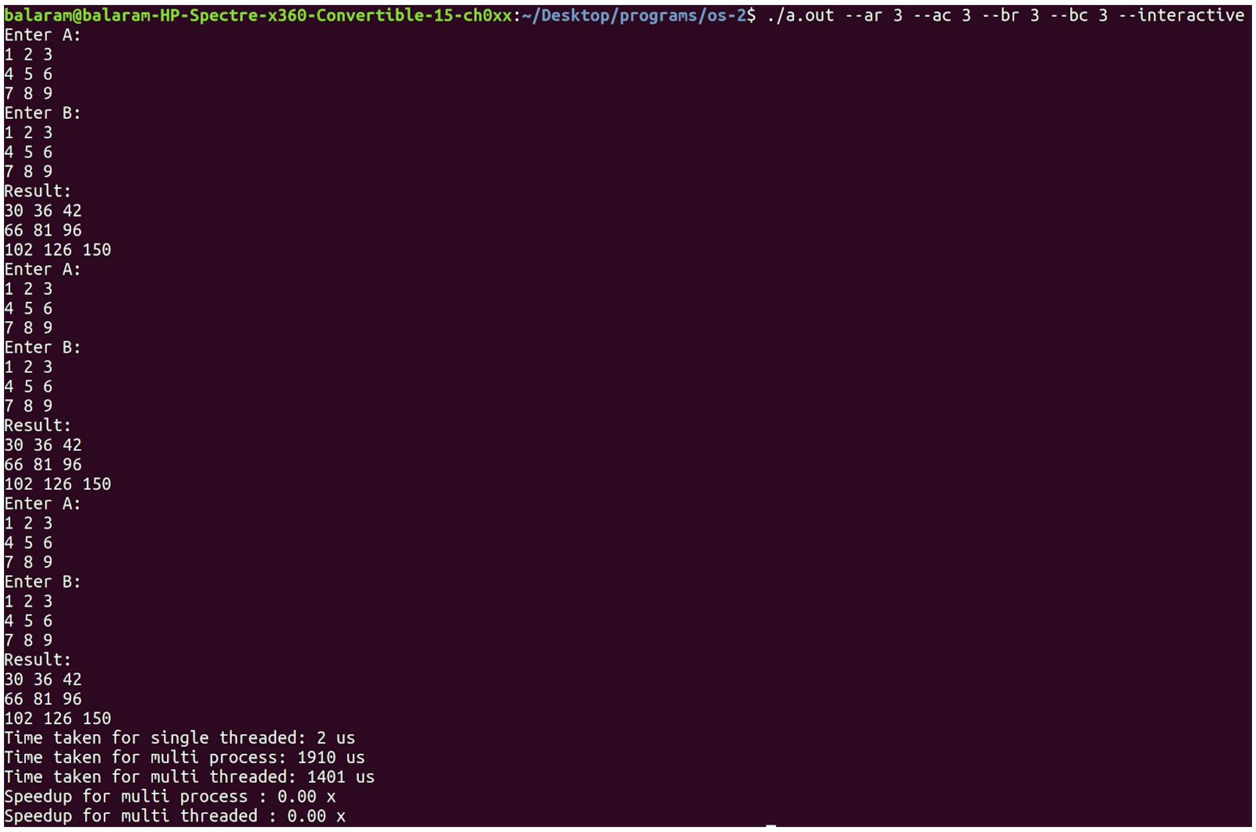
```
balaram@balaram-HP-Spectre-x360-Convertible-15-ch0xx:~/Desktop/programs/os-2$ ./a.out --ar 500 --ac 700 --br 700 --bc 500
Time taken for single threaded: 822330 us
Time taken for multi process: 266464 us
Time taken for multi threaded: 252432 us
Speedup for multi process : 3.09 x
Speedup for multi threaded : 3.26 x
```

Test Case-6



```
balaram@balaram-HP-Spectre-x360-Convertible-15-ch0xx:~/Desktop/programs/os-2$ ./a.out --ar 1000 --ac 1000 --br 1000 --bc 1000
Time taken for single threaded: 5470492 us
Time taken for multi process: 1701968 us
Time taken for multi threaded: 1653869 us
Speedup for multi process : 3.21 x
Speedup for multi threaded : 3.31 x
```

Test Case-7



```
balaram@balaram-HP-Spectre-x360-Convertible-15-ch0xx:~/Desktop/programs/os-2$ ./a.out --ar 3 --ac 3 --br 3 --bc 3 --interactive
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: 1910 us
Time taken for multi threaded: 1401 us
Speedup for multi process : 0.00 x
Speedup for multi threaded : 0.00 x
```

Observation

By observing the test cases which are run on my terminal for small size of the matrix the single threaded is performing much faster than the multi threaded program and multi process program. This is due to, in the low input case in single thread the loop is small and multiplication is completed faster and in multi threaded case there will be context switches happening and also locks in the case of count increment and due to the switches happening the time is being increased and hence the total time in case of small inputs is more and in case of multi process child process are being created and switching from one child to another child takes time which causes an overhead in the time. We can observe that as the input size is growing the speed of single thread kept on increasing and the multithreading time and the multi process time became comparatively low with single thread, which tells the significance in case of multithreading and multi process programming each process or thread takes up their part does the computation giving a sense of faster computing as it does.Here multithreading is faster because the context switch between the threads is easier and faster when compared to multi process. In multi threading a small amount of memory copying is required, in multiprocess parent should transfer the data with it to child complete.In memory sharing between threads they all lie in a process which makes it easy here where as is in the case of multi process we need to access the shared memory region which makes it slow.The matrix multiplication with use of multithreading makes it much faster in the case of longer and big inputs.