CIS620 **PROJECT-1** **FALL2022**

NAME: -JAGADEESH PORALLA
CSU ID: - 2826049
LOGIN ID: - japorall@grail.eecs.csuohio.edu
COURSE: - Advanced Operating Systems

CODE DESCRIPTION: -

1)As mentioned in the handout, I have created 4 .c files with the same names provided and used both the start_timer() and stop_timer() as utility functions which are used in every program and defined in timing.c

2) I have used the global variable(elapsed_sec) in timing.c with static keyword as a specifier to include the flavor of object-oriented language to hide information.

3) My ep.c evaluates the time taken to create the process with the dynamic memory allocation size specified from the command Line using the utility functions defined in timing program when compiled and linked together to create executable files ep_crt, whereas et.c does the same creating et_crt file.

4) The para_mm.c will do the parallel matrix multiplication according to the threads number provided from user. This program is compiled and linked to timing.c creating para_mm for calculating the time taken according to the number of threads.

5) In order to allocate or clear the memory before calling the start_timer(), I have used calloc() which helps in finding the affect of dynamic memory allocation as well.

6) Both my child process and thread write a set of characters to array of size mentioned in order to create a job for them to compare the time taken. The size of calloc is user defined and unit was mentioned as 1024(1K-Byte =1024bytes).

7) At the end, I have used separate compilation to create a makefile which describe the dependency among these files and

use the timestamp to build the executable files when their dependencies are updated by user.

8) In order to give a hard time to the process, we have written something(same) in the child side of both thread and process using char array of memory size. This helped us do the same work on using both and calculate the time taken.

EXPERIENCES IN TESTING/DEBUGGING:-

Most of the testing I have done was use of printf to find the compile and execution status before linking it to other files.
I have declared main in the timing.c initially to find the result of program since it was 0, if it is run alone before debugging.
All the other three programs were tested by printing each output to make sure it works exactly.

timing.c :-

1) While testing the timing.c program, I have noticed there is just a change in microseconds which results in dropping of least number while adding(float) with seconds after conversion using 10^6 as dividend. I have tried to debug this is many ways but at last I settled with double as type for elapsed_sec. Here I leant how float is stored in a 32-bit with 24-bit precession.

ep.c:-

1) In the for loop used for filling character array , I have got a warning for i+4096 as an increment since the result of this expression is unused. I have assigned it to the same variable I, In order to make the compiler think I have used it making it i=i+4096.

et.c

1) Since I have used the same way to copy on write as ep.c , I have assigned the increment to same variable to remove the warning in the for loop. But, along with that I have encountered an issue in passing the arguments to function declared in thread creation. I have both size and array that must be sent while I can send Just one. For solving this, I have passed array as an argument and used sizeof(), to use it in the function.
2) After passing character array to the function, I have used pointer of pointers to store the array and deference it to find the value present inside the address [use of *(char**), which was really tough to find out

.

para_mm.c:-

This part of project took long to create the threads which shares the portion of matrix multiplication according to the number of threads. The function called with thread id as an argument needs lot more information to do the whole calculation. For this I have defined the three arrays at the start of the program and number of threads using a global variable to make it visible from the main and Matrix_multiply function. Even then, defining the beginning and ending of specific portion was hard which I solved using start and end as variable with their own formula according to the thread number. Use of pthread_join(), helped to find out when to stop the timer after completing the whole job.

PROJECT STATUS: -

My project completely works according to the mentioned parameters and gives the results accordingly.
The command line arguments will make the executables work according to the filename and parameters passed.

EXPERIMENTS AND RESULTS: -

1) For both ep_crt ,et_crt I have tried different memory sizes like 1024, 2048, 4096, 8192, 16383 whose results are compared in the below table in seconds.

SIZE(Kbytes)	**PROCESS**	**THREADS**
1024	0.000579sec	0.000112sec
2048	0.000304sec	0.000085sec
4096	0.000235sec	0.000019sec
8192	0.000318sec	0.000028sec
16384	0.000593sec	0.000055sec

SCREENSHOT: -

```
japorall@spirit:~/CIS620$ make
make: Nothing to be done for 'all'.
japorall@spirit:~/CIS620$ ./ep_crt 1024
the time taken to create process for size 1024 K-Bytes is 0.000579sec
japorall@spirit:~/CIS620$ ./et_crt 1024
the time taken to create thread for size 1024 K-Bytes is 0.000112
japorall@spirit:~/CIS620$ ./ep_crt 2048
the time taken to create process for size 2048 K-Bytes is 0.000304sec
japorall@spirit:~/CIS620$ ./et_crt 2048
the time taken to create thread for size 2048 K-Bytes is 0.000085
japorall@spirit:~/CIS620$ ./ep_crt 4096
the time taken to create process for size 4096 K-Bytes is 0.000235sec
japorall@spirit:~/CIS620$ ./et_crt 4096
the time taken to create thread for size 4096 K-Bytes is 0.000019
japorall@spirit:~/CIS620$ ./ep_crt 8192
the time taken to create process for size 8192 K-Bytes is 0.000318sec
japorall@spirit:~/CIS620$ ./et_crt 8192
the time taken to create thread for size 8192 K-Bytes is 0.000028
japorall@spirit:~/CIS620$ ./ep_crt 16384
the time taken to create process for size 16384 K-Bytes is 0.000593sec
japorall@spirit:~/CIS620$ ./et_crt 16384
the time taken to create thread for size 16384 K-Bytes is 0.000055
japorall@spirit:~/CIS620$ vim et.c
japorall@spirit:~/CIS620$ make
```

As per the above table, thread takes less time as it is a lightweight process. Since threads take less time to create or terminate by sharing the resources like code, data, resources. By increasing the size of memory, the time used is decreased until it reaches the length of the memory page (4096), later again [for 8192) it is increased since it goes to a new memory page.

2)All the programs were executed alone by using main () to call other functions for checking the working of time finding, process and thread creation.

3)In the matrix multiplication, I have tested using 1,2,4,8,16 as the number of threads which created 320x320, 2of 160x160, 4 of 80x80, 8 of 40x40 and 16 of 20x20 matrixes respectively. This was made possible after

many attempts to exactly formulate the outermost for loop with a defined start and end of rows in 1st matrix.
4) The time taken to compute matrix multiplication was lowered simultaneously by increasing the number of threads until it reached 16 threads. I really have no exact idea for why it has gone up for splitting into 16 parts. (i.e 20x20 matrix).

```
japorall@spirit:~/CIS620$ ./ para_mm 1
-bash: ./: Is a directory
japorall@spirit:~/CIS620$ ./para_mm 1
finished part 1
time taken to do using 1 threads is 0.070778sec
japorall@spirit:~/CIS620$ ./para_mm 2
finished part 1
finished part 2
time taken to do using 2 threads is 0.035086sec
japorall@spirit:~/CIS620$ ./para_mm 3
finished part 1
finished part 2
finished part 3
time taken to do using 3 threads is 0.024628sec
japorall@spirit:~/CIS620$ ./para_mm 3make
finished part 1
finished part 2
finished part 3
time taken to do using 3 threads is 0.024437sec
japorall@spirit:~/CIS620$ make
make: Nothing to be done for 'all'.
japorall@spirit:~/CIS620$ ./para_mm 1
finished part 1
time taken to do using 1 threads is 0.070097sec
japorall@spirit:~/CIS620$ ./para_mm 2
finished part 1
finished part 2
time taken to do using 2 threads is 0.035391sec
japorall@spirit:~/CIS620$ ./para_mm 4
finished part 1
finished part 2
finished part 3
finished part 4
time taken to do using 4 threads is 0.017323sec
japorall@spirit:~/CIS620$ ./para_mm 8
finished part 1
finished part 2
finished part 3
finished part 4
finished part 5
finished part 6
finished part 7
finished part 8
time taken to do using 8 threads is 0.013742sec
japorall@spirit:~/CIS620$ ./para_mm 16
finished part 1
finished part 2
finished part 3
finished part 4
finished part 5
finished part 6
finished part 7
finished part 8
finished part 9
finished part 10
finished part 11
finished part 12
finished part 13
finished part 14
finished part 15
finished part 16
time taken to do using 16 threads is 0.011992sec
japorall@spirit:~/CIS620$
```

5) For makefile , I have used “all” as a target with ep_crt , et_crt , para_mm as its dependencies. The required commands were added some using macros and the others directly in the command section. I have added -lpthread and pthread also in order to compile the thread and matrix programs.

In this whole project process, I have used lot of time for debugging and testing since the class notes helped me to create all the basic code needed for every program using the constrains defined in the handout including the global variable, memory size, copy on write and dynamic memory allocation.