Name: Sakshi Vinayak Kitturkar
CSU ID: 2860273
Grail Login: sakittur

Project1

- In this project we are asked to write four C files which are etime.c, et.c, ep.c, para_mm.c.
- The file etime.c contains a function etime() which returns the time taken since the last call to etime(). And must implement it by using gettimeofday () and static local variable.
- The file et.c and etime.c are compiled and linked together to build the executable file et. In this file we need to call pthread_create/ pthread_join() to evaluate the time for thread creation and deletion.
- The file ep.c and etime.c are compiled and linked together to generate the executable file ep. In this we must invoke fork()/ waitpid() to measure the process creation and deletion time.
- The file para_mm.c and etime.c are compiled and linked together to build the executable para_mm.c in this we need to measure the computation time for multiplying two 400 * 400 matrices. we must split the task to a few threads and each thread just handles a portion of the matrix.
- We must write a makefile to describe the dependency among these files and to build the three executable files et, ep and para_mm.

etime.c

```c
#include <stdlib.h>
#include <sys/time.h>
#include <stdio.h>

struct timeval start; // structure to hold start time

float etime (){
struct timeval stop; // structure to hold stop time
float time_taken;
gettimeofday (&stop, NULL);
time_taken=(stop.tv_sec - start.tv_sec) + (stop.tv_usec - start.tv_usec) / 1000000.0; // update the start time for the next measurement
start = stop;
return time_taken;
}
```

- I have included the necessary header files needed which are: ‘stdlib.h’, ‘sys/time.h’ and ‘stdio.h’.
- The file etime will have two functions which I have taken as start and stop.
- We are using one global variable ‘struct timeval start’ to store the start time for timing measurements.
- We have a function defined as ‘etime ()’ that returns the elapsed time since the last call to the function.
- Inside the etime function we declare a local variable ‘ struct timeval stop’ to store the stop time for timing measurements.
- Then using a formula which calculates the time_taken in seconds with microsecond precision.
- Next it updates the start time for the next measurement to be the same as the stop time.
- And returns the calculated time_taken.

et.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<pthread.h>

extern float etime(); // linking of etime
int number;
char* buffer;
char* argument;
// Thread function to perform memory access based on command line argument.
void*function (void *arg) {
    if(strcmp(argument,"-a") ==0) {
      for (int i = 0; i<number*1024; i= i+4096) {
         buffer[i] = 20;
      }
   }
}
int main(int argc, char *argv[]){
   // check if the command line arguments is not equal to 3
if (argc != 3) {
printf("/et -a size\n") ; // print usage information
exit (1);
}

float time_taken;
int number=atoi (argv[2]); //convert the second command line argument to an integer
int unitK=1024;
argument=argv [1];
char* buffer = (char*) calloc(number, unitK);// allocate memory for the buffer array
pthread_t task; // declared a pthread variable for the task
```

```
etime();


if (strcmp(argument,"-b") ==0) {
  for (int i = 0; i < number*unitK; i=i+4096) {
    buffer[i] = 20;
  }
}


pthread_create(&task, NULL,function, NULL); // create a new thread to execute the function
pthread_join (task, NULL); // wait for the thread to finish execution
time_taken = etime();
printf("Total time taken:%f seconds\n", time_taken);// print the total time taken


}
```

- I have included the necessary header files like ‘stdio.h’, ‘stdlib.h’, ‘string.h’, and ‘pthread.h’ to facilitate standard I/O operations, memory allocation, string manipulation and multi-threading.
- We have externally linked the ‘etime()’ function to our code which is used to measure the time taken for memory access operation.
- There are three variables which I have declared as global variable those are ‘number’, ‘buffer’, ‘argument’.
- Thread function it will perform memory access operations based on the command-line argument passed to the program. The argument is ‘-a’ which access the memory in chunks the function doesn’t return anything.
- We need to call pthread_create()/ pthread_join() to evaluate the time for thread creation/deletion.
- In the Main function it checks if the correct number of command-line argument are provided if not it prints the statement and exits. It will initialize variables including ‘number’, ‘unitK’, and ‘argument’ based on command line argument. I have allocated the memory to the buffer array using ‘calloc()’ based on the number of elements and the unit size. Memory access operations are performed based on the command line ‘-a’ and ‘-b’ if the argument -b is accessed a new thread is created to execute the function(). if its ‘-a’ the main thread waits for the thread to finish using ‘pthread_join ()’ and the total time taken will be printed.

Output:

```
sakittur@spirit:~$ cd cis620
sakittur@spirit:~/cis620$ cd project1
sakittur@spirit:~/cis620/project1$ gcc -o et et.c etime.c -lpthread
sakittur@spirit:~/cis620/project1$ ./et -b 0
Total time taken:0.000140 seconds
sakittur@spirit:~/cis620/project1$ ./et -b 1024
Total time taken:0.000308 seconds
sakittur@spirit:~/cis620/project1$ ./et -b 8192
Total time taken:0.000181 seconds
sakittur@spirit:~/cis620/project1$ ./et -b 16384
Total time taken:0.000211 seconds
sakittur@spirit:~/cis620/project1$ ./et -b 32768
Total time taken:0.000283 seconds
```

```
sakittur@spirit:~/cis620/project1$ ./et -a 0
Total time taken:0.000289 seconds
sakittur@spirit:~/cis620/project1$ ./et -a 1024
Total time taken:0.000295 seconds
sakittur@spirit:~/cis620/project1$ ./et -a 8192
Total time taken:0.000135 seconds
sakittur@spirit:~/cis620/project1$ ./et -a 16384
Total time taken:0.000149 seconds
sakittur@spirit:~/cis620/project1$ ./et -a 32768
Total time taken:0.000162 seconds
```

ep.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<sys/time.h>

extern float etime(); //externally linking the etime() function
int main(int argc, char *argv[]){
```

```
if (argc != 3)
{
printf("/ep -a size\n");// prints usage information if incorrect number of arguments provided
exit (1);// exit the program
}

float time_taken;
etime();
int unitK=1024;
int number = atoi (argv [2]);
char* buffer = (char*) calloc (number, unitK);// allocate memory for the buffer array
pid_t cpid;// variable to hold child process ID
if (strcmp(argv[1], "-b") == 0){
for (int i = 0; i < number*unitK; i= i+4096) {
buffer [i] = 20;
   }
}
if ((cpid==fork()) == 0){
//this is child
if (strcmp(argv[1], "-a") == 0) {
for (int i = 0; i < number*unitK; i= i+4096) {
buffer [i] = 20;
}
}
exit(0);// exit the child process
}else{
//this is parent
// wait for the child process to finish execution
waitpid (cpid, NULL, 0);
time_taken = etime();
printf("Total time taken is: %f\n", time_taken); // print the total time taken
}
return 0;
}
```

- I have included the necessary header files for this program.
- In this we had to invoke fork ()/waitpid () to measure the process creation/deletion time.
- We have externally linked the ‘etime ()’ function to our code.
- The program starts from the main function it will first check whether the correct number of command-line arguments is provided if its not correct it prints usage information and exits the program.
- Memory is allocated for the buffer array using’calloc ()’ with the specified number of elements and size.
- The code checks the command line argument (‘argv [1]’) to determine the type of memory access operation. If the argument is ‘-b’ it will access memory in chunks filling the buffer with the value 20.
- The code will fork a child process using ‘fork ()’ if the fork () returns 0 indicating it’s the child process the child executes the corresponding memory access operation on the command line argument. If it’s the parent process it waits for the child process to finish execution using ‘waitpid ()’.
- After the child process completes and the parent process resumes the time_taken is calculated using etime () to measure the total time taken for memory access operation.

Output:

```
sakittur@spirit:~/cis620/project1$ ./ep -b 0
Total time taken is: 0.000272
sakittur@spirit:~/cis620/project1$ ./ep -b 1024
Total time taken is: 0.000731
sakittur@spirit:~/cis620/project1$ ./ep -b 8192
Total time taken is: 0.000427
sakittur@spirit:~/cis620/project1$ ./ep -b 16384
Total time taken is: 0.000696
sakittur@spirit:~/cis620/project1$ ./ep -b 32768
Total time taken is: 0.001205
```

```
sakittur@spirit:~/cis620/project1$ gcc -o ep ep.c etime.c
sakittur@spirit:~/cis620/project1$ ./ep -a 0
Total time taken is: 0.000615
sakittur@spirit:~/cis620/project1$ ./ep -a 1024
Total time taken is: 0.002999
sakittur@spirit:~/cis620/project1$ ./ep -a 8192
Total time taken is: 0.004785
sakittur@spirit:~/cis620/project1$ ./ep -a 16384
Total time taken is: 0.009498
sakittur@spirit:~/cis620/project1$ ./ep -a 32768
Total time taken is: 0.019521
```

para_mm.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <math.h>
#include <pthread.h>
#include<semaphore.h>

#define MS 400 // the size of the square matrices
int n;  // number of threads
extern float etime();//externally linking the etime() function

int MA[MS][MS];// declare the first matrix
int MB[MS][MS];// declare the second matrix
int MC[MS][MS];// declare the result matrix

sem_t sem;//semaphore for synchronization
void *worker(void *arg) {
    long tid = (long)arg; //thread id
    int start_row =(MS / n)*tid;// calculate starting row for this thread
    int end_row =(MS / n)* (tid +1) -1;// calculate ending row for this thread

    for (int i = start_row; i < end_row; i++) {
```

```
        for (int j = 0; j < MS; j++) {
            MC[i][j] = 0;
            for (int k = 0; k < MS; k++) {
                MC[i][j] += MA[i][k] * MB[k][j];
            }
        }
    }
    sem_post(&sem);//signal completion of work by this thread
}
int main(int argc, char *argv[]) {
int start_row;// start row for matrix initialization
int end_row;// end row for matrix initialization
    sem_init(&sem,0,0);
    if (argc != 2) {
        printf(" not arguments:\n");
        return 1;
    }
 n = atoi(argv[1]);
    pthread_t threads[n];// declare array to hold thread id
    for (int i = start_row; i < end_row; i++) {
        for (int j = 0; j < MS; j++) {
            MA[i][j] = 0;
            }
         }
            for (int i = start_row; i < end_row; i++) {
        for (int j = 0; j < MS; j++) {
            MB[i][j] = 0;
            }
         }
etime();
    // create specified number of worker threads
    for (long i = 0; i < n; i++) {
        pthread_create(&threads[i], NULL, worker, (void *)i);
    }
    // wait for all threads to finish execution
    for (long i = 0; i< n; i++){
        sem_wait(&sem);
```

```
    }
    float time_taken = etime();
    printf("Total time taken: %.6f seconds\n", time_taken);
    return 0;
}
```

- I have included necessary header file like ‘stdio.h’, ‘stdlib.h’, ‘sys/time.h’, ‘math.h’, ‘pthread.h’, and ‘semaphore.h’ to facilitate standard I/O operations, memory allocation, time-related functions, mathematical operations, multi-threading support and semaphore-based synchronization.
- MS defines the size of the square matrices which is 400.
- The global variables that I have declared are ‘n’ which represents the number of threads to be used for matrix multiplication. ‘MA’, ‘MB’, ‘MC’ which is two-dimensional array representing the input matrices ‘A’, ‘B’ and the result matrix ‘C’. ‘sem’ which is used as semaphore synchronization between threads.
- Worker function: in this each thread executes the ‘worker’ function which will perform a portion of matrix multiplication. It will calculate the starting and ending rows to process based on the thread ID. Then the thread will iterate over assigned rows and performs matrix multiplication using nested loops.
- Main function: in main it checks weather the correct number of command line arguments is provided if not then printing a message of incorrect. It will initialize variables based on the command-line argument, specifically the number of the threads use ‘n’. the matrixes ‘MA’ and ‘MB’ are initialized with zeros. Semaphore ‘sem’ is initialized. Time measurement begins with the call to ‘etime’ worker threads are created using ‘pthread_create’, each executing the ‘worker’ function. The main thread waits for all worker threads to finish execution using ‘sem_wait’. then time_taken for matrix multiplication is measured and the total time taken for matrix multiplication is printed.

Output:

```
sakittur@spirit:~/cis620/project1$ gcc -o para_mm para_mm.c etime.c -lpthread
sakittur@spirit:~/cis620/project1$ ./para_mm 1
Total time taken: 0.271810 seconds
sakittur@spirit:~/cis620/project1$ ./para_mm 2
Total time taken: 0.140373 seconds
sakittur@spirit:~/cis620/project1$ ./para_mm 4
Total time taken: 0.071591 seconds
sakittur@spirit:~/cis620/project1$ ./para_mm 8
Total time taken: 0.070141 seconds
sakittur@spirit:~/cis620/project1$ ./para_mm 16
Total time taken: 0.074968 seconds
sakittur@spirit:~/cis620/project1$
```

makefile

```
CC = gcc
LIBS = -lpthread

all: et ep para_mm

et: et.o etime.o
	$(CC) -o et et.o etime.o $(LIBS)

ep: ep.o etime.o
	$(CC) -o ep ep.o etime.o $(LIBS)

para_mm: para_mm.o etime.o
	$(CC) -o para_mm para_mm.o etime.o $(LIBS)

et.o: et.c
	$(CC) -c et.c

ep.o: ep.c
	$(CC) -c ep.c

para_mm.o: para_mm.c
	$(CC) -c para_mm.c

etime.o: etime.c
```

```
    $(CC) -c etime.c
clean:
    rm *.o et ep para_mm
```

- CC = gcc : sets the compiler to GNU compiler collection.
- LIBS = -lpthread : specifies the '-lpthread' flag to link the pthread library.
- 'all' : specifies the default target which builds all executables.
- Each target ('et.c', 'ep.c', 'para_mm.c') is associated with a rule that specifies how to link object files.
- Each source file ('et.c', 'ep.c', 'para_mm.c', 'etime.c')has a rule that specifies how to compile it into an object file.
- The compilation command (' $(CC) -c) is followed by the source file name.
- The 'clean' specifies how to clean up the project directory by removing object files ('*.o') and executables ('et.c', 'ep.c', 'para_mm.c')

- During the development process I extensively tested my code to ensure its correctness. Throughout the development process I encountered and got various bugs and issues. I carefully analyzed error messages examined variables values and stepped through the code to understand its execution flow and identify potential issues. This project helped me learn about the threads and processes. And my project status is 'WORKS'.