OS ASSIGNMENT-2 REPORT

Lab3- Matrix Multi

: Adarsh Anand : 2003101 :

: Prof - Dr Sharad Sinha : TA - Prachi Kashikar

QUESTIONS

Q) Write a C program that computes the product of two matrices, using multiple threads for execution.

● Your program must conform to the following prototype:

my_matrix_multiply -a a_matrix_file.txt -b b_matrix_file.txt -t thread_count

where the -a and -b parameters specify input files containing matrices and thread_count is the number of threads to use in your strip decomposition.

The input matrix files are text files having the following format.

The first line contains two integers: rows columns. Then each line is an element in row major order. Lines that begin with “#” should be considered comments and should be ignored. Here is an example context from a matrix file:

```
3 2
# Row 0
0.711306
0.890967
# Row 1
0.345199
0.380204
# Row 2
0.276921
0.026524
```

This matrix has 3 rows and 2 columns and contains comment lines showing row boundaries in the row-major ordering.

● You need to generate random matrices in this format and do the argument parsing necessary so that the prototype shown above works properly.

● Your program will need to print out the result of A * B where A is contained in the file passed via the -a parameter and B is contained in the file passed via the -b parameter. It must print the product in the same format (with comments indicating rows) as the input matrix files: rows and columns on the first line, each element in row-major order on a separate line.

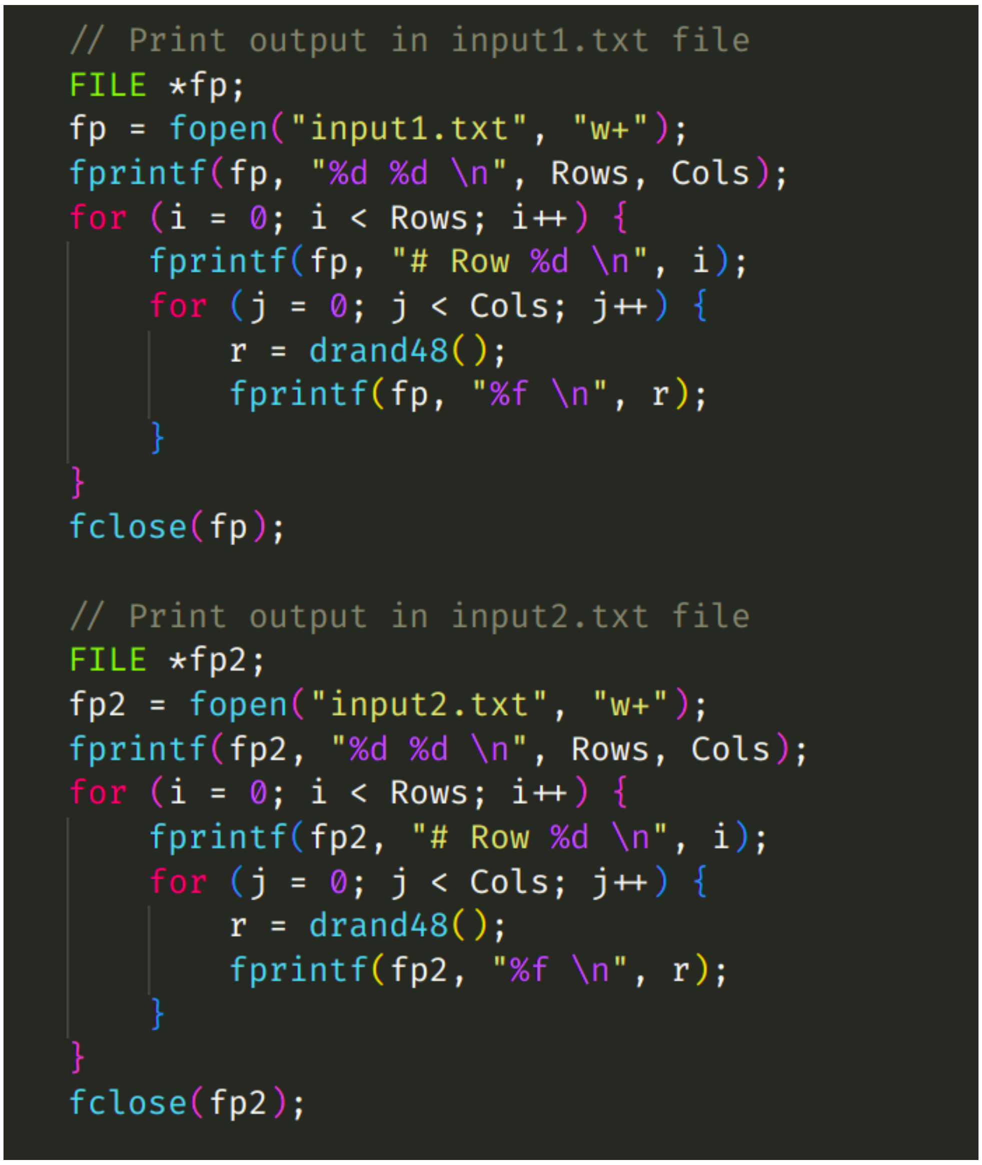
● You also have to check the time required to complete the computation with and without multiple threads.

Some C codes for threading, random matrix generation and timer are provided along with this manual, for your reference.

ANSWERS

1. Creating matrices and creating 2 files
2. Reading 2 generated files and parsing matrix from it
3. Creating threads and doing matrix multiplication
4. Finding out time for all the operations
5. Varying the threads and getting the execution time
6. Plotting graphs using matplotlib and seaborn for the threads vs execution time

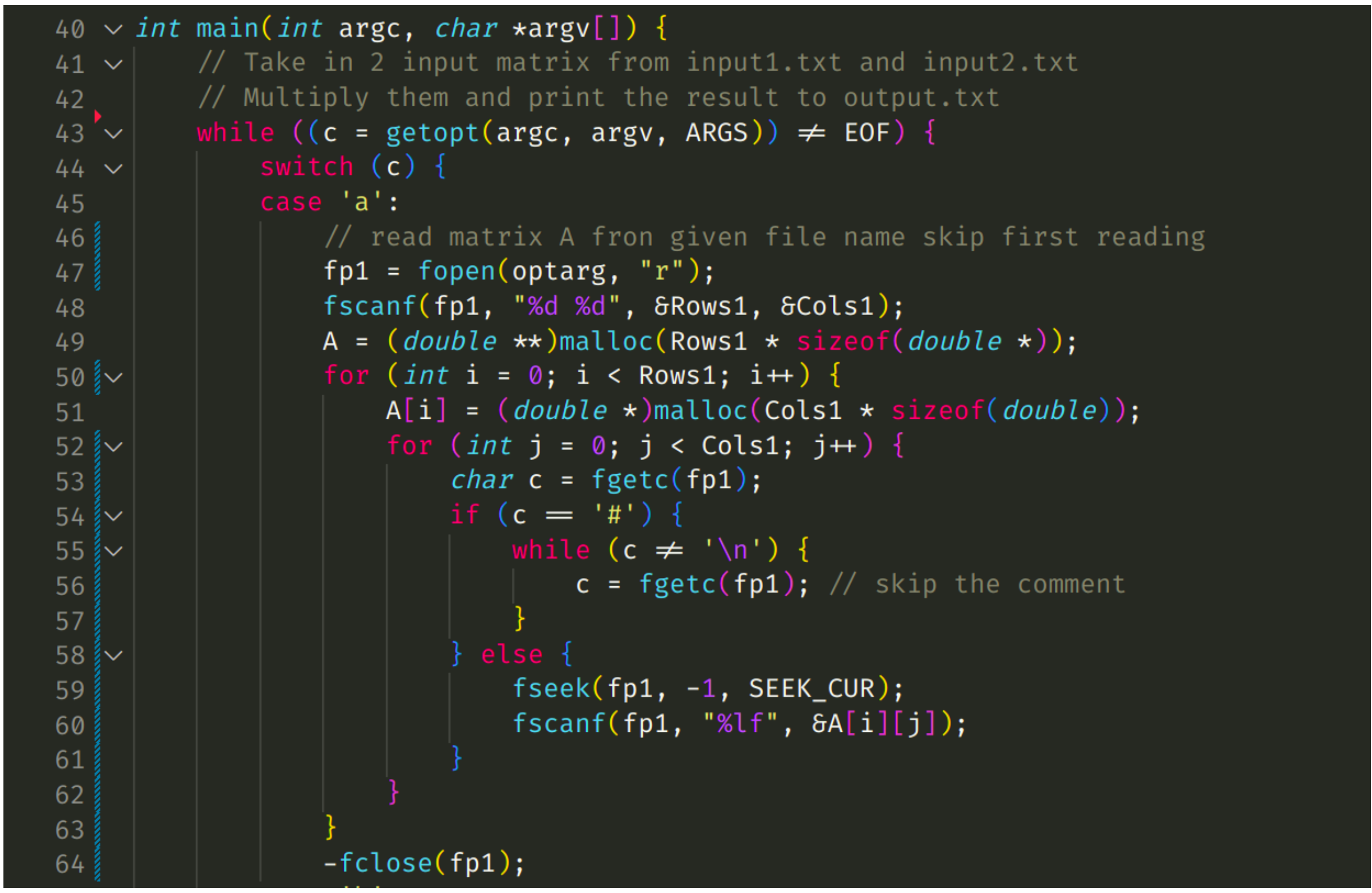
1. Reading files and outputting



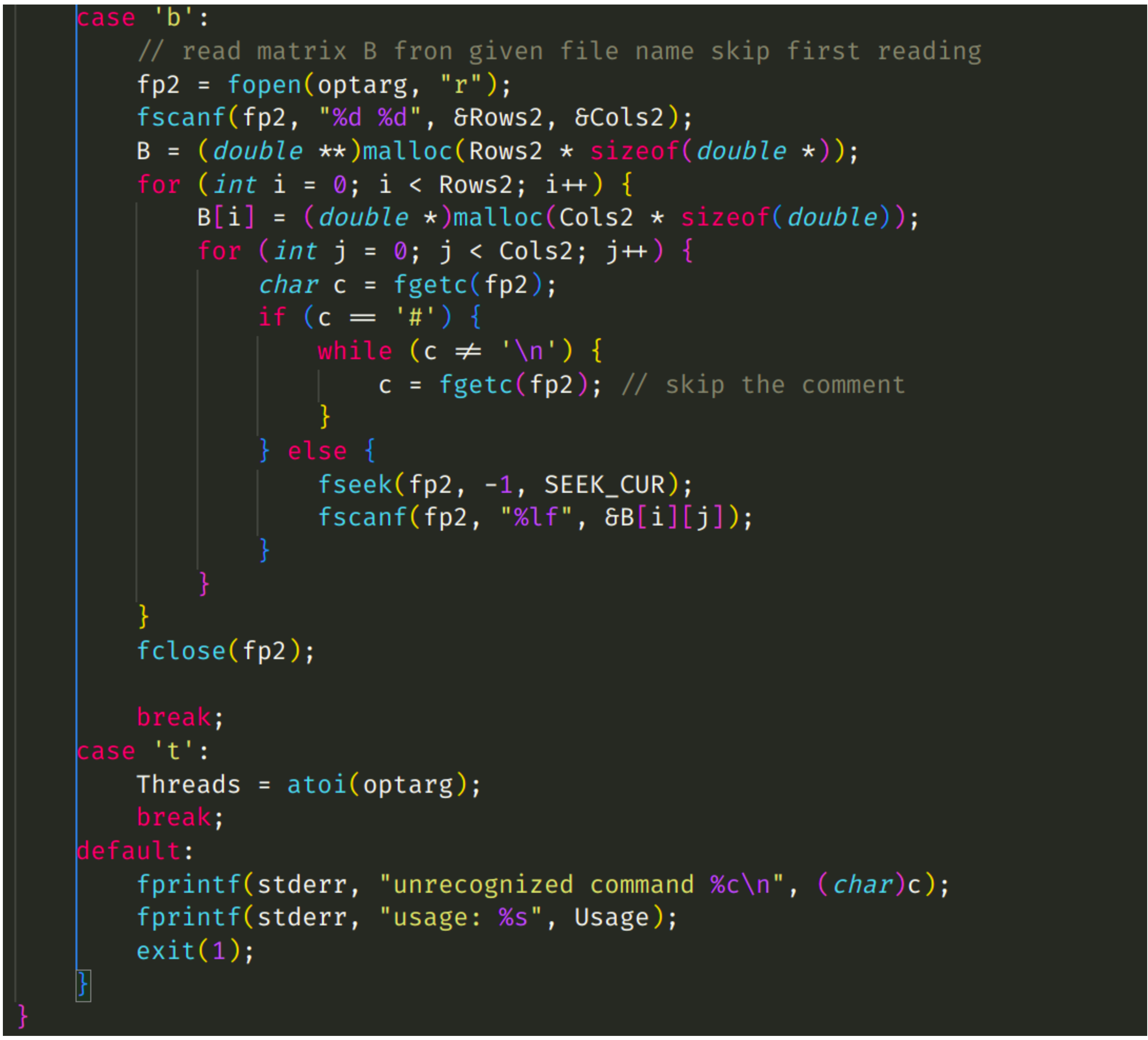
```c
// Print output in input1.txt file
FILE *fp;
fp = fopen("input1.txt", "w+");
fprintf(fp, "%d %d \n", Rows, Cols);
for (i = 0; i < Rows; i++) {
    fprintf(fp, "# Row %d \n", i);
    for (j = 0; j < Cols; j++) {
        r = drand48();
        fprintf(fp, "%f \n", r);
    }
}
fclose(fp);

// Print output in input2.txt file
FILE *fp2;
fp2 = fopen("input2.txt", "w+");
fprintf(fp2, "%d %d \n", Rows, Cols);
for (i = 0; i < Rows; i++) {
    fprintf(fp2, "# Row %d \n", i);
    for (j = 0; j < Cols; j++) {
        r = drand48();
        fprintf(fp2, "%f \n", r);
    }
}
fclose(fp2);
```

2. Parsing matrix



```c
int main(int argc, char *argv[]) {
    // Take in 2 input matrix from input1.txt and input2.txt
    // Multiply them and print the result to output.txt
    while ((c = getopt(argc, argv, ARGS)) != EOF) {
        switch (c) {
        case 'a':
            // read matrix A fron given file name skip first reading
            fp1 = fopen(optarg, "r");
            fscanf(fp1, "%d %d", &Rows1, &Cols1);
            A = (double **)malloc(Rows1 * sizeof(double *));
            for (int i = 0; i < Rows1; i++) {
                A[i] = (double *)malloc(Cols1 * sizeof(double));
                for (int j = 0; j < Cols1; j++) {
                    char c = fgetc(fp1);
                    if (c == '#') {
                        while (c != '\n') {
                            c = fgetc(fp1); // skip the comment
                        }
                    } else {
                        fseek(fp1, -1, SEEK_CUR);
                        fscanf(fp1, "%lf", &A[i][j]);
                    }
                }
            }
            -fclose(fp1);
```

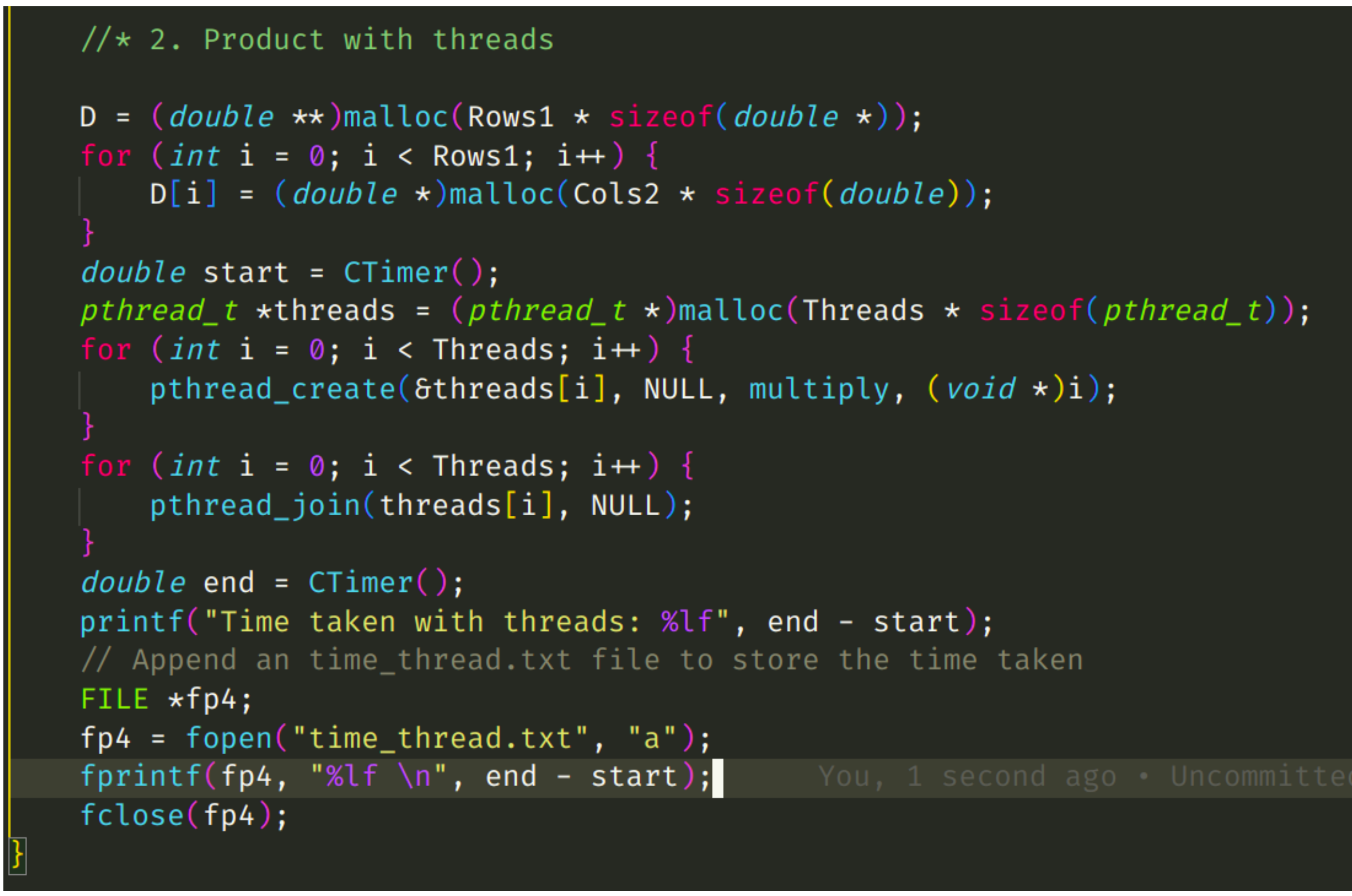


```
    case 'b':
        // read matrix B fron given file name skip first reading
        fp2 = fopen(optarg, "r");
        fscanf(fp2, "%d %d", &Rows2, &Cols2);
        B = (double **)malloc(Rows2 * sizeof(double *));
        for (int i = 0; i < Rows2; i++) {
            B[i] = (double *)malloc(Cols2 * sizeof(double));
            for (int j = 0; j < Cols2; j++) {
                char c = fgetc(fp2);
                if (c == '#') {
                    while (c != '\n') {
                        c = fgetc(fp2); // skip the comment
                    }
                } else {
                    fseek(fp2, -1, SEEK_CUR);
                    fscanf(fp2, "%lf", &B[i][j]);
                }
            }
        }
        fclose(fp2);

        break;
    case 't':
        Threads = atoi(optarg);
        break;
    default:
        fprintf(stderr, "unrecognized command %c\n", (char)c);
        fprintf(stderr, "usage: %s", Usage);
        exit(1);
    }
}
```

3.

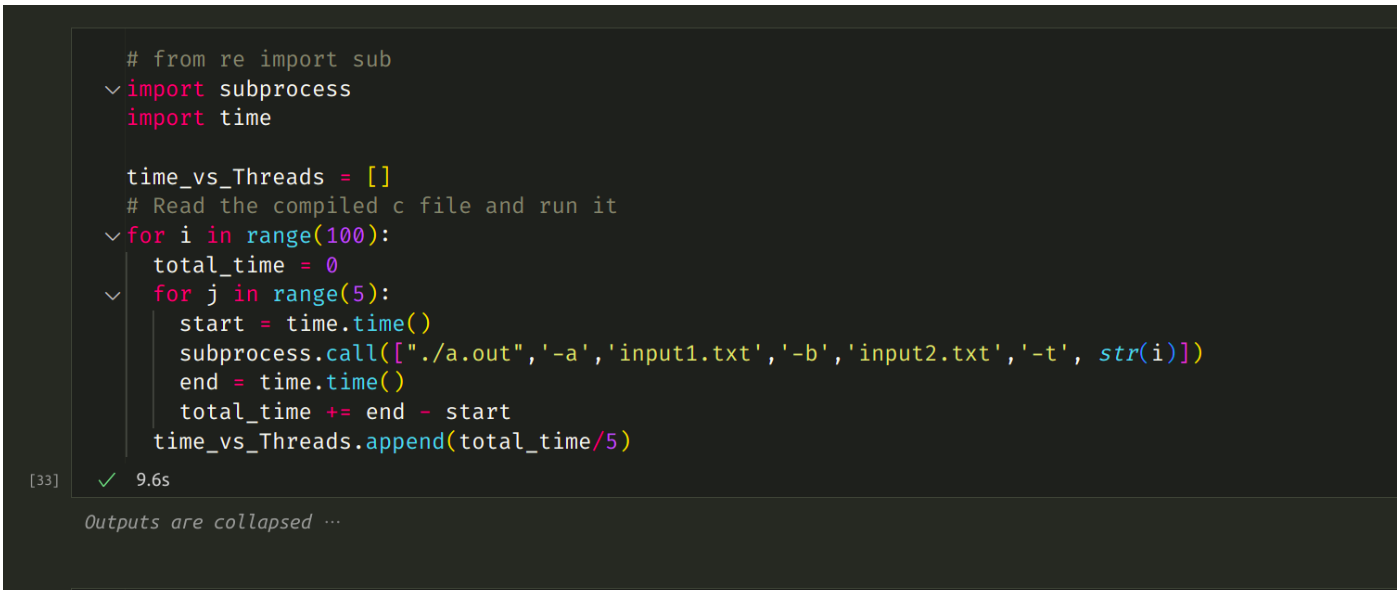
4. Product using threads



```c
    //* 2. Product with threads

    D = (double **)malloc(Rows1 * sizeof(double *));
    for (int i = 0; i < Rows1; i++) {
        D[i] = (double *)malloc(Cols2 * sizeof(double));
    }
    double start = CTimer();
    pthread_t *threads = (pthread_t *)malloc(Threads * sizeof(pthread_t));
    for (int i = 0; i < Threads; i++) {
        pthread_create(&threads[i], NULL, multiply, (void *)i);
    }
    for (int i = 0; i < Threads; i++) {
        pthread_join(threads[i], NULL);
    }
    double end = CTimer();
    printf("Time taken with threads: %lf", end - start);
    // Append an time_thread.txt file to store the time taken
    FILE *fp4;
    fp4 = fopen("time_thread.txt", "a");
    fprintf(fp4, "%lf \n", end - start);      You, 1 second ago • Uncommitte
    fclose(fp4);
}
```

5. Simulating - Time vs Threads



```python
# from re import sub
import subprocess
import time

time_vs_Threads = []
# Read the compiled c file and run it
for i in range(100):
  total_time = 0
  for j in range(5):
    start = time.time()
    subprocess.call(["./a.out",'-a','input1.txt','-b','input2.txt','-t', str(i)])
    end = time.time()
    total_time += end - start
  time_vs_Threads.append(total_time/5)
```

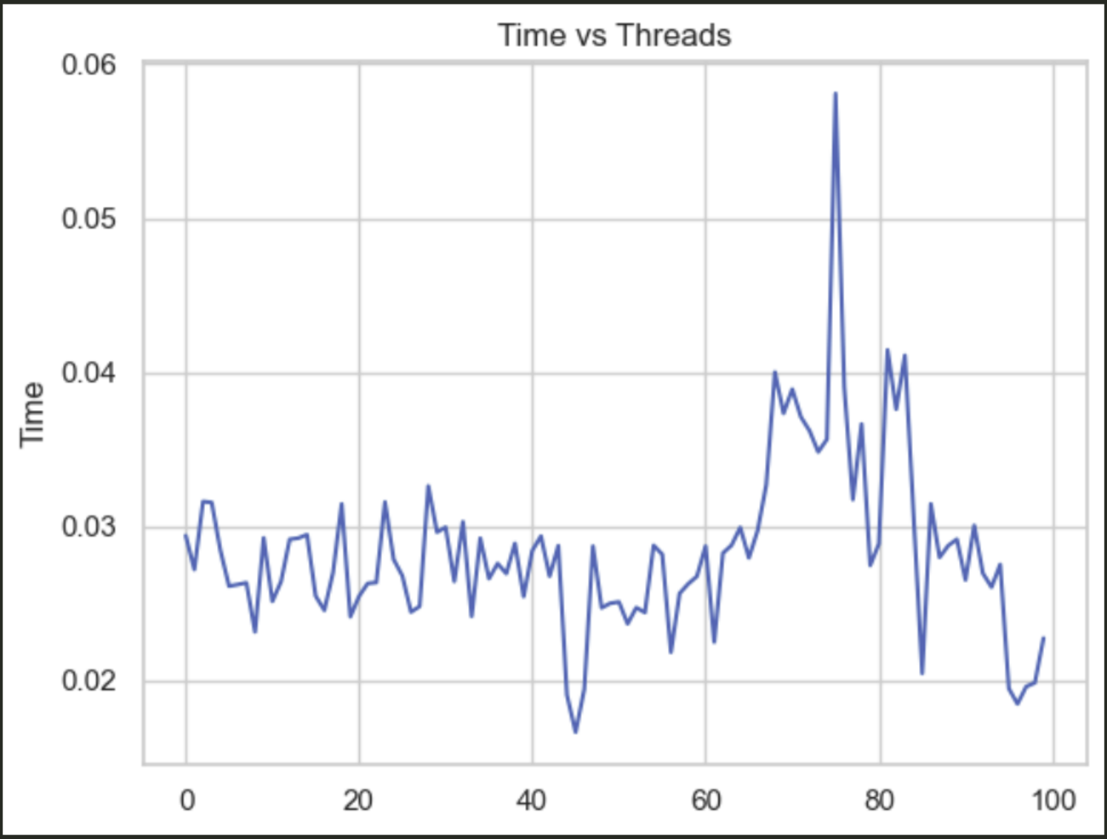
```
[33]  ✓  9.6s

Outputs are collapsed ···
```

```
# Plot the graph for time vs threads
plt.plot(time_vs_Threads)
plt.title('Time vs Threads')
plt.ylabel('Time')
```

```
[61]  ✓ 0.4s
```

```
Text(0, 0.5, 'Time')
```



REFERENCES

- Dr Sharad Sir Slides
- C Programs given in .zip file
- Input-output system calls in C | Create, Open, Close, Read, Write - GeeksforGeeks
- Linux man pages
- Bash manual
- Wait System Call in C - GeeksforGeeks

END OF REPORT