

**Faculty of Engineering & Technology**  
**Electrical & Computer Engineering Department**  
**Operating Systems**

**Task#1**

**Report**

**Experimenting Matrix-Multiplication**  
**Throw Different Operations**

---

**Prepared by:**  
**Name:** Ody Shbayeh

**Number:** 1201462

**Instructor:** Dr.Abdelsalam-sayyad

**Section:** 2

**Date :** 30/11/2023

## **Abstract :**

The aim of the Task is to go through all the possible methods to obtain the most and lowest time possible when working with the processes like matrix-multiplication using the normal multiplication way and by dividing the processes and by threading them.

## **Theory :**

### **1. Matrix-multiplication :**

Matrix-multiplication is one of the most used mathematical theories to solve most of the array's problems and for developing the 2D and 3D programs that needs to calculate the sum of two or three vectors and get the resultant vector or determining the accurate position of an object .

The way the matrix multiplication works is that it takes two separate array's and it multiplies the first element in the first row in the first matrix and multiply it with the first column in the second matrix that by the first index in the first matrix is multiplied by the first index in the second matrix and summed by that every index is multiplied by the index it matches it's position.

### **2. Time-measurement :**

Time is an a important factor to improve the performance of a program or a working component in a system or circuit , when working with large number of processes we need to implement methods to finish the needed tasks in the shortest time consumed so that we can processes with more complex problems .

The way that Time-measurement works in the systems that it starts counting the Time from the current second then it runs the process after period of time we get the time after the process end's running then we the time after the process ends and subtract it from the time we get before the process begin by that we can get the time period that the process takes.

### **3. Distributing the Processes :**

Distributing processes is an effective way to end the job very fast and deal with complex problems ,cause one core can handle a specified number of processes but when distributing that large number of processes among a good number of cores in the CPU the time it takes to end these processes end's much more faster by that we can launch cores to work on a specified number of operations together and end's it in a short time period .

For some programs like games or other programs that have huge number of processes needs to be executed the method of executing the program or these number of processes one-by-one isn't efficient or effective by that we have a method of distributing the work needed to be done in parallel like what GPU's do and the theory behind this is that we create number of threads to handle these operations or by using the way of forking the processes to parents and child's so that we can end the job faster that the original way of executing it normally.

## Procedure :

### 1. Creating matrixes and filling them :

As our Task included that we want to create matrixes and fill them with a specified sequence to experiment the matrix multiplication process.

The method used in my code that I make a matrix that have the specified sequence : 1201462 which is my ID card number in an array called "stdnum" And created another one to hold the other sequence that made by "myID\*yearof birth" that is named by "stdnumyear" and have the sequence " 2405326924" that makes the product of my ID "1201462"\*"2002" which is my year of birth

After that I make nested loops for each of the matrixes to fill them using a counter that reaches number 7 which is the length of my ID to reset the filling and fill the whole matrix the same thing domne to the other matrix to fill it with the specified sequence from before in the following figure is a screenshot for my code that creates and fills the matrix :

```
116 //odyshbayeh-1201462
117 int main() {
118     int matrix_id[100][100];
119     int matrix_q[100][100];
120     int result[100][100];
121     int option;
122     int stdnum [7] = {1,2,0,1,4,6,2};
123     int stdnumyear [10] = {2,4,0,5,3,2,6,9,2,4};
124     int counter1 = 0;
125     int counter2 = 0;
126     for (int i=0; i<100;i++)
127     {
128         for(int j=0; j<100;j++)
129         {
130             matrix_id[i][j]=stdnum[counter1];
131             if (counter1 == 7)
132             {
133                 counter1=0;
134             }else {
135                 counter1++;
136             }
137         }
138     }
139     for (int i=0; i<100;i++)
140     {
141         for(int j=0; j<100;j++)
142         {
143             matrix_q[i][j]=stdnumyear[counter2];
144             if (counter2 == 10)
145             {
146                 counter2=0;
147             }else{
148                 counter2++;
149             }
150         }
151     }
152     struct timeval start, end;
153     do {
154         printf("odyshbayeh-1201462\n\n");
155         printf("1 normal matrix multiplication\n");
156         printf("2 matrix multiplication using threads\n");
157     } while(1);
158 }
```

Activate Windows  
Go to Settings to activate Win

C Tab Width: 8 Ln 116, Col 21

## 2. Making a UI for the user :

It's commonly known that to get to full experience for the experiment is to make a UI for the user to launch the tasks.

The method for making the UI is very simple we define a int called choice and scan the user choice in a do-while loop then making that choice runs in a switch statement that have cases of the functions that we want the console to execute aka "matrix-multiplication-methods"

In the header of the do-while loop I put down some printing statements to let the user knows which number to choose to execute the method of multiplying And the loop breaks when the user chooses the number 0 which is the faulty condition for the while loop that makes it to break and end the program.

```
166 //odyshbayeh-1201462
167 printf("5 run all four methods\n");
168 printf("0 exit\n");
169 printf("please enter a number from the options above : ");
170 scanf("%d", &option);
171
172
173
174 switch (option) {
175     case 1:
176         gettimeofday(&start, NULL);
177         normal_multiplication(result, matrix_id, matrix_q);
178         gettimeofday(&end, NULL);
179         printf("the prosses took %lf milliseconds using normal multiplication\n", timediff(start, end));
180         printf("\n\n");
181         break;
182
183     case 2:
184         for (int thread_count = 2; thread_count <= 8; thread_count *= 2) {
185             gettimeofday(&start, NULL);
186             matrix_multiplication_threads(thread_count, matrix_id, matrix_q, result, 0);
187             gettimeofday(&end, NULL);
188             printf("number of threads used for the multiplication : %d ", thread_count);
189             printf("the prosses took %lf milliseconds\n", timediff(start, end));
190
191         } printf("\n\n");
192         break;
193
194     case 3:
195         for (int process_count = 2; process_count <= 8; process_count *= 2) {
196             gettimeofday(&start, NULL);
197             matrix_multiplication_process(process_count, matrix_id, matrix_q, result);
198             gettimeofday(&end, NULL);
199             printf("number of processes used for the multiplication : %d ", process_count);
200             printf("the prosses took %lf milliseconds\n", timediff(start, end));
201
202         }
203         printf("\n\n");
204         break;
205     case 4:
206         for (int thread_count = 2; thread_count <= 8; thread_count *= 2) {
207             gettimeofday(&start, NULL);
208             matrix_multiplication_threads(thread_count, matrix_id, matrix_q, result, 1);
209             gettimeofday(&end, NULL);
210             printf("number of detached threads used for the multiplication : %d ", thread_count);
211             printf("the prosses took %lf milliseconds\n", timediff(start, end));
212
213
214 }
215 printf("\n\n");
216 break;
217
218 case 5:
219     gettimeofday(&start, NULL);
220     normal_multiplication(result, matrix_id, matrix_q);
221     gettimeofday(&end, NULL);
222     printf("the prosses took %lf milliseconds using normal multiplication\n", timediff(start, end));
223
224     for (int thread_count = 2; thread_count <= 8; thread_count *= 2) {
225         gettimeofday(&start, NULL);
226         matrix_multiplication_threads(thread_count, matrix_id, matrix_q, result, 0);
227         gettimeofday(&end, NULL);
228         printf("number of threads used for the multiplication : %d ", thread_count);
229         printf("the prosses took %lf milliseconds\n", timediff(start, end));
230
231     }
232
233     for (int process_count = 2; process_count <= 8; process_count *= 2) {
234         gettimeofday(&start, NULL);
235         matrix_multiplication_process(process_count, matrix_id, matrix_q, result);
236         gettimeofday(&end, NULL);
237         printf("number of process used for the multiplication : %d ", process_count);
238         printf("the prosses took %lf milliseconds\n", timediff(start, end));
239
240     }
241     for (int thread_count = 2; thread_count <= 8; thread_count *= 2) {
242         gettimeofday(&start, NULL);
243         matrix_multiplication_threads(thread_count, matrix_id, matrix_q, result, 1);
244         gettimeofday(&end, NULL);
245         printf("number of detached threads used for the multiplication : %d ", thread_count);
246         printf("the prosses took %lf milliseconds\n", timediff(start, end));
247
248     }
249     printf("\n\n");
250     break;
251 case 0:
252     option = 0;
253     break;
254 }
255 while (option != 0);
256 return 0;
257 }
```

## For the cases in the switch statements :

In the case number 1 :

We get the current time to be able to calculate the time difference before launching the program and after it after that we call the function called `normal_multiplication` that takes the `matrix_id` and the `matrix_q` and the result matrix defined in the main as parameters to do the matrix multiplication.

The function `normal_multiplication` is a nested loop to go through all the indexes inside the `matrix_id` and the `matrix_q` and putting the resultant in the matrix result.

```
17 void normal_multiplication(int result[100][100], int matrix_id[100][100], int matrix_q[100][100]) {
18     for (int i = 0; i < 100; ++i) {
19         for (int j = 0; j < 100; ++j) {
20             result[i][j] += matrix_id[i][j] * matrix_q[i][j];
21         }
22     }
23 }
```

After that we get the current time again and call the function called "`timediff`" That takes two parameters the start and the end that refers to the times we get previously and get the result of them then we print the time consumed for the operation.

```
112
113 double timediff(struct timeval start, struct timeval end) {
114     return (double)((end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1000.0);
115 }
116
```

For the case number 2 :

The method of measuring the time remains the same but we create a loop to send the number of threads we want to the function `matrix_multiplication_threads` that takes the thread count taken from the loop and the Detached parameter which is 0 in this case to do a normal multiplication using joint threads and of course the matrixes we want to execute the operations on :

```
64 //qxbxbxbxb-1201462
65 void matrix_multiplication_threads(int thread_count, int matrix_id[100][100], int matrix_q[100][100], int result[100][100], int detached) {
66     pthread_t t[thread_count];
67     pthread_attr_t attr;
68
69     if (detached) {
70         pthread_attr_init(&attr);
71         pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
72     }
73
74     threadstructure thread_data[thread_count];
75
76     for (int i = 0; i < thread_count; ++i) {
77         thread_data[i].sr = i * (100 / thread_count);
78         thread_data[i].er = (i + 1) * (100 / thread_count);
79         for (int j = 0; j < 100; ++j) {
80             for (int k = 0; k < 100; ++k) {
81                 thread_data[i].matrix_id[j][k] = matrix_id[j][k];
82                 thread_data[i].matrix_q[j][k] = matrix_q[j][k];
83             }
84         }
85     }
86     for (int i = 0; i < thread_count; ++i) {
87         if (detached) {
88             if (pthread_create(&t[i], &attr, worker_thread, &thread_data[i]) != 0) {
89                 fprintf(stderr, "pthread_create failed\n");
90                 exit(0);
91             }
92         } else {
93             if (pthread_create(&t[i], NULL, worker_thread, &thread_data[i]) != 0) {
94                 fprintf(stderr, "pthread_create failed\n");
95                 exit(0);
96             }
97         }
98     }
99 }
```

```

8
9 typedef struct threadstructure{
10     int matrix_id[100][100];
11     int matrix_q[100][100];
12     int result[100][100];
13     int sr;
14     int er;
15 } threadstructure;

```

For the parameter taken it checks if the threads is detached or not in our case it's not so it skips the first if statement then it creates a threadstructure which is a predefined structure that have a 3 matrixes and a start and end int to split the matrixes fills then in the wotking arrays defined in the threadstructure then work with them.

The method of splitting the array's is by setting the int sr to the first index if the int I was 0 and set the er to 50 which is the array size defined by the number of threads passed in the argument previously this makes the threading works for dividing the matrix depending on the thread count after that we fill the array's in the threadstructure after that we enter's a new loop with a condition of rechecking for Detached and calls the worker thread that actually do the matrix multiplication then enter's another loop to fill the result's of the multiplying to the result array then fill it and then we do the measuring time and print the elapsed time for the process

```

//odvshhaye-1201462
void* worker_thread(void* arg) {
    threadstructure* data = (threadstructure*)arg;

    for (int i = data->sr; i < data->er; ++i) {
        for (int j = 0; j < 100; ++j) {
            data->result[i][j] = 0;
            for (int k = 0; k < 100; ++k) {
                data->result[i][j] += data->matrix_id[i][j] * data->matrix_q[i][j];
            }
        }
    }
}

98
99 if (detached) {
100     usleep(10000);
101 }
102 for (int i = 0; i < thread_count; ++i) {
103     for (int j = thread_data[i].sr; j < thread_data[i].er; ++j) {
104         for (int k = 0; k < 100; ++k) {
105             result[j][k] = thread_data[i].result[j][k];
106         }
107     }
108 }
109

```

For case number 3 :

In this case we go threw another way of Distributing the work of matrix multiplication by using the function Fork() that basically makes child's for the parent operation and when we get the child id we can simply make the matrix multiplication we want to achieve and we also have the wait Function that wait's for all child's to end their processes then makes the parent's do their processes after that we measure the time it takes it to do it for the whole operation using the same method :

```

38 void matrix_multiplication_process(int process_count, int matrix_id[100][100], int matrix_q[100][100], int result[100][100]) {
39
40     pid_t p_num[process_count];
41
42     for (int i = 0; i < process_count; ++i) {
43         int first_line = i * (100 / process_count);
44         int last_line = (i + 1) * (100 / process_count);
45
46         pid_t child_pid = fork();
47
48         if (child_pid == 0) {
49             for (int row = first_line; row < last_line; ++row) {
50                 for (int col = 0; col < 100; ++col) {
51                     result[row][col] = 0;
52                     for (int k = 0; k < 100; ++k) {
53                         result[row][col] += matrix_id[row][k] * matrix_q[k][col];
54                     }
55                 }
56             }
57             exit(1);
58         }
59     }
60     for (int i = 0; i < process_count; ++i) {
61         int status;
62         waitpid(p_num[i], &status, 0);
63     }
64 }

```

For case number 4 :

In this case we go through the same steps for the case number 2 but the difference we have that now we are making detached threads and passing 1 in the calling in statement to work with the function :

```

8
9 typedef struct threadstructure{
10     int matrix_id[100][100];
11     int matrix_q[100][100];
12     int result[100][100];
13     int sr;
14     int er;
15 } threadstructure;
16
17 //sdypbhavsh-1201462
64 void matrix_multiplication_threads(int thread_count, int matrix_id[100][100], int matrix_q[100][100], int result[100][100], int detached) {
65     pthread_t t[thread_count];
66     pthread_attr_t attr;
67
68     if (detached) {
69         pthread_attr_init(&attr);
70         pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
71     }
72
73     threadstructure thread_data[thread_count];
74
75     for (int i = 0; i < thread_count; ++i) {
76         thread_data[i].sr = i * (100 / thread_count);
77         thread_data[i].er = (i + 1) * (100 / thread_count);
78         for (int j = 0; j < 100; ++j) {
79             for (int k = 0; k < 100; ++k) {
80                 thread_data[i].matrix_id[j][k] = matrix_id[j][k];
81                 thread_data[i].matrix_q[j][k] = matrix_q[j][k];
82             }
83         }
84     }
85
86     for (int i = 0; i < thread_count; ++i) {
87         if (detached) {
88             if (pthread_create(&t[i], &attr, worker_thread, &thread_data[i]) != 0) {
89                 fprintf(stderr, "pthread_create failed\n");
90                 exit(0);
91             }
92         } else {
93             if (pthread_create(&t[i], NULL, worker_thread, &thread_data[i]) != 0) {
94                 fprintf(stderr, "pthread_create failed\n");
95                 exit(0);
96             }
97         }
98     }
99     if (detached) {
100         usleep(10000);
101     }
102     for (int i = 0; i < thread_count; ++i) {
103         for (int j = thread_data[i].sr; j < thread_data[i].er; ++j) {
104             for (int k = 0; k < 100; ++k) {
105                 result[j][k] = thread_data[i].result[j][k];
106             }
107         }
108     }
109 }

```



```

//adysbhavsh-1201462
void* worker_thread(void* arg) {
    threadstructure* data = (threadstructure*)arg;

    for (int i = data->sr; i < data->er; ++i) {
        for (int j = 0; j < 100; ++j) {
            data->result[i][j] = 0;
            for (int k = 0; k < 100; ++k) {
                data->result[i][j] += data->matrix_id[i][j] * data->matrix_q[i][j];
            }
        }
    }
}

```

For case number 5 :

it's simply a case that run's all the cases from previous to also see if there's an change in the time measurement and the algorithm for it is the same for every case before :

1. Get the time and save it
2. Call the function with the right parameters for it
3. Get the time and save it
4. Call the timediff function to get the elapsed time
5. Print the results

Finally for case number 6 :

It's simply a case to exit the program by setting the choice option to 0 which breaks the do-while loop.

### 3. Libraries used in the project :

```

1 //adysbhavsh-1201462
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <sys/time.h>
5 #include <sys/wait.h>
6 #include <unistd.h>
7 #include <pthread.h>

```

Stdio.h is the well known input-output library.

Stdlib.h is the library that have the other libraries

The previous two are in already in all the projects when creating the project

Unistd.h is the library that make it able to make the fork() function

Pthread.h is the library that makes it able to create threads and work with them

Sys/wait.h is a library that makes the system have the function wait() that makes the system able to wait for the parents to wait for all the child to finish

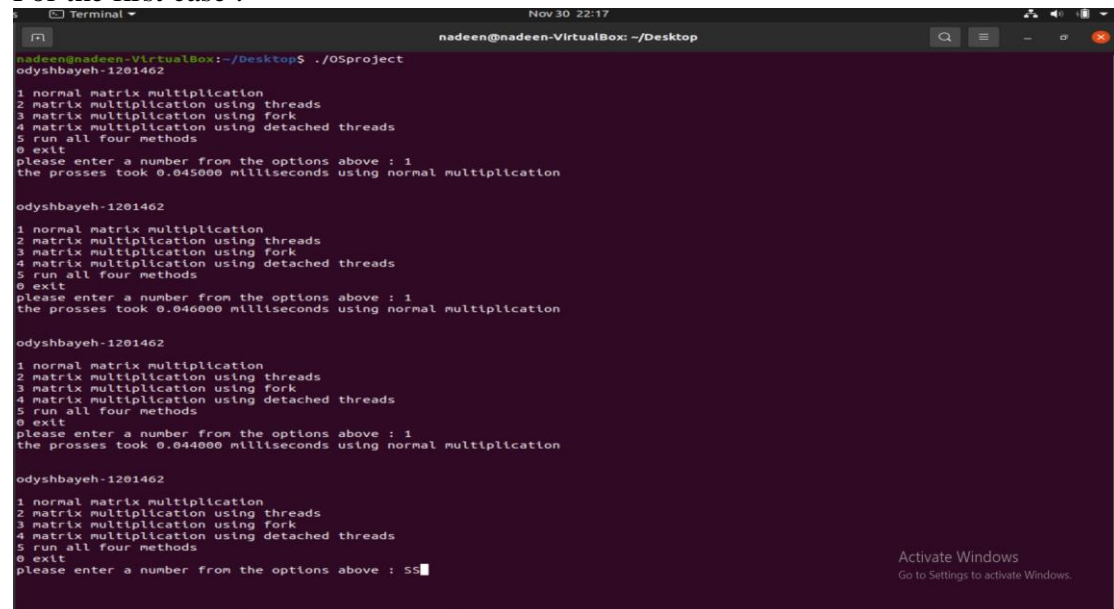
Sys/time.h is the library that have the current time to measure the time lapse for the processes in the Task

These libraries and these operations are running only on UNIX and doesn't run on the windows API Operating System

## Results :

After running the previous cases for three times for each case I came up with these results :

For the first case :



```
nadeen@nadeen-VirtualBox: ~/Desktop
odyshbayeh-1201462
1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 1
the prosses took 0.045000 milliseconds using normal multiplication

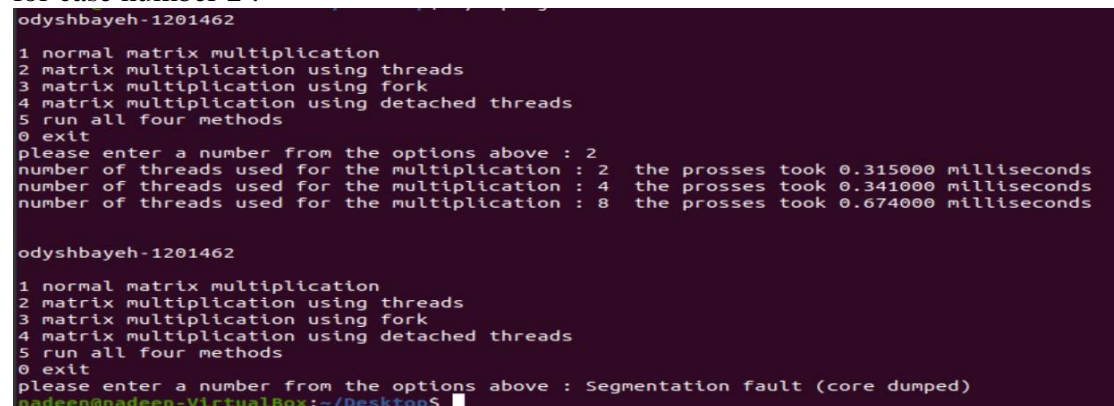
odyshbayeh-1201462
1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 1
the prosses took 0.046000 milliseconds using normal multiplication

odyshbayeh-1201462
1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 1
the prosses took 0.044000 milliseconds using normal multiplication

odyshbayeh-1201462
1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 5
Segmentation fault (core dumped)
```

I've the virtual-box for running the UNIX OS on my compute from before casue my sister had it before me that have the name : nadeen-shbayeh and ID : 1180517 and I took some help from her doing this program

for case number 2 :



```
odyshbayeh-1201462
1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 2
number of threads used for the multiplication : 2 the prosses took 0.315000 milliseconds
number of threads used for the multiplication : 4 the prosses took 0.341000 milliseconds
number of threads used for the multiplication : 8 the prosses took 0.674000 milliseconds

odyshbayeh-1201462
1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 5
Segmentation fault (core dumped)
```

Note \*\* I had this problem of the code being dumped on most of the cases still don't know why but some times it works some times it doesn't .

### For case number 3 :

```
1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 3
number of processes used for the multiplication : 2 the proseses took 0.124000 milliseconds
number of processes used for the multiplication : 4 the proseses took 0.150000 milliseconds
number of processes used for the multiplication : 8 the proseses took 2.708000 milliseconds

odyshbayeh-1201462

1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 3
number of processes used for the multiplication : 2 the proseses took 0.103000 milliseconds
number of processes used for the multiplication : 4 the proseses took 0.150000 milliseconds
number of processes used for the multiplication : 8 the proseses took 0.284000 milliseconds

odyshbayeh-1201462

1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 3
number of processes used for the multiplication : 2 the proseses took 0.102000 milliseconds
number of processes used for the multiplication : 4 the proseses took 0.257000 milliseconds
number of processes used for the multiplication : 8 the proseses took 0.399000 milliseconds

odyshbayeh-1201462

1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : █
```

Activate Windows  
Go to Settings to activate Windows.

### For case number 4 :

```
1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 4
number of detached threads used for the multiplication : 2 the proseses took 10.509000 milliseconds
number of detached threads used for the multiplication : 4 the proseses took 11.121000 milliseconds
number of detached threads used for the multiplication : 8 the proseses took 10.645000 milliseconds

odyshbayeh-1201462

1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 4
number of detached threads used for the multiplication : 2 the proseses took 10.360000 milliseconds
number of detached threads used for the multiplication : 4 the proseses took 10.299000 milliseconds
number of detached threads used for the multiplication : 8 the proseses took 10.681000 milliseconds

odyshbayeh-1201462

1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 4
number of detached threads used for the multiplication : 2 the proseses took 10.436000 milliseconds
number of detached threads used for the multiplication : 4 the proseses took 10.343000 milliseconds
number of detached threads used for the multiplication : 8 the proseses took 10.668000 milliseconds

odyshbayeh-1201462

1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : █
```

Activate Windows  
Go to Settings to activate Windows.

### For case number 5 :

Note \*\* I ran it once cause it get's dumped for no reason

```
odyshbayeh-1201462

1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : 5
the proseses took 0.043000 milliseconds using normal multiplication
number of threads used for the multiplication : 2 the proseses took 0.253000 milliseconds
number of threads used for the multiplication : 4 the proseses took 0.355000 milliseconds
number of threads used for the multiplication : 8 the proseses took 0.696000 milliseconds
number of process used for the multiplication : 2 the proseses took 0.160000 milliseconds
number of process used for the multiplication : 4 the proseses took 0.287000 milliseconds
number of process used for the multiplication : 8 the proseses took 5.227000 milliseconds
number of detached threads used for the multiplication : 2 the proseses took 11.325000 milliseconds
number of detached threads used for the multiplication : 4 the proseses took 13.255000 milliseconds
number of detached threads used for the multiplication : 8 the proseses took 10.864000 milliseconds

odyshbayeh-1201462

1 normal matrix multiplication
2 matrix multiplication using threads
3 matrix multiplication using fork
4 matrix multiplication using detached threads
5 run all four methods
0 exit
please enter a number from the options above : █
```

## Conclusion :

From this Task I've learned a lot about CPU work mechanism and how can we reduce the time consumed running a program or processing a operation , I came though the fork() function and saw and understood how it works and how effective it was to reduce the time and also for the algorithm of distributing the work using threads .

The previous report was the answer for the first question from the task :

1- the task is about matrix multiplication.

Part was covered

2- you need to generate your own matrices for performance comparison.

A- Generate a 100X100 matrix of integers, composed of your student number repeated until the matrix is filled.

Part was covered

B- Generate another 100X100 matrix matrix of integers, composed of (your student number \* your birth year) repeated until the matrix is filled.

Part was covered

3- compare the following **four** approaches. Measure the time it takes to complete the program in each case.

A- the naive approach, a program that does not use any child processes or threads.

Case 1 in the program.

B- a program that uses multiple child processes running in parallel. Try different numbers of child processes and compare the outcome.

Case 3 in the program.

C- a program that uses multiple joinable threads running in parallel. Try different numbers of threads and compare the outcome.

Case 2 in the program.

D- **a program that uses multiple detached threads running in parallel. Try different numbers of threads and compare the outcome. Is it possible to measure the time in this approach?**

Case 4 in the program. // yes it can be measured but it's not efficient since it takes a lot of time to do it.

- 4- Document all your experiments, and provide the results in an appropriate table. Submit this documentation along with your code. **Your code may be in one file or multiple files, as you see fit.**

For the code I'll attach a .c file with this PDF document

For the table I considered the results in pictures but I'll add another one here having some of the results :

Exp	Normal	Joinable threads	Detached threads	Fork()	Number of child's	Number of joinable threads	Number of detached threads
1	0.045 ms	0.025 ms	10.5 ms	0.012 ms	2	2	2
1	0.046 ms	0.023 ms	11.12 ms	0.011 ms	4	4	4
1	0.044 ms	0.025 ms	10.6 ms	2.7 ms	8	8	8

- 5- What is the proper/optimal number of child processes or threads? Justify your answer.

The optimal number of child's or threads was 4 child's/threads

The reason behind that is the CPU I have is INTEL I7 that have 7 cores and a fast response to the operations so when executing the program 4 cores had that same amount of processes to finish them which IG was the optimal case for them.