

BAHRIA UNIVERSITY,

# (Karachi Campus)

*Department ofsoftware Engineering*

ASSIGNMENT #: 3 — Fall 2022

### COURSE TITLE: System Programming

Class: BSE - 5(B)

### Course Instructor: Engr. Rizwan Fazal Due Date: 13-January-2023

COURSE CODE: CEN-449

### Shift: Morning

Max. Marks: 10 Points

# ASSIGNMENT #. 3

## Name: M Muaz Shahzad

Enrolment#:02-131202-081

## Class: BSE 5B

Software Engineering Department

Bahria University, Karachi Campus

Single Thread:

Code:

#include<stdio.h>

#include<pthread.h>

#include<unistd.h>

#include<stdlib.h>

#include <algorithm>

#include <chrono>

#include <iostream>

#include<vector>

using namespace std;

using namespace std::chrono;

#define mrow 2

#define mcol 2

#define vrow 2

#define vcol 1

#define MAX\_THREAD 2

int mat[mrow][mcol];

int vec[vrow][vcol];

int res[mrow][vcol];

void\* MatrixMultiply(void\* arg) {

for (int i = 0; i < mrow; i++) {

for (int j = 0; j < vcol; j++) {

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

res[i][j] += mat[i][k] \* vec[k][j];

}

}

}

}

int main()

{

auto start = high\_resolution\_clock::now();

pthread\_t threads;

for (int i = 0; i < mrow; i++) {

for (int j = 0; j < mcol; j++) {

mat[i][j] = rand() % 10;

}

}

for (int i = 0; i < vrow; i++) {

for (int j = 0; j < vcol; j++) {

vec[i][j] = rand() % 10;

}

}

cout << endl

<< "Matrix" << endl;

for (int i = 0; i < mrow; i++) {

for (int j = 0; j < mcol; j++)

cout << mat[i][j] << " ";

cout << endl;

}

cout << endl

<< "Vector:" << endl;

for (int i = 0; i < vrow; i++) {

for (int j = 0; j < vcol; j++) {

cout << vec[i][j] << "\n";

}

}

pthread\_t tread;

int ret;

ret = pthread\_create(&threads, NULL, MatrixMultiply, NULL);

if (!ret) {

cout << "Resultant vector" << endl;

for (int i = 0; i < mrow; i++) {

for (int j = 0; j < vcol; j++) {

cout << "" << res[i][j];

cout << endl;

}

}

}

auto stop = high\_resolution\_clock::now();

Output:

Multithreading:

Code:

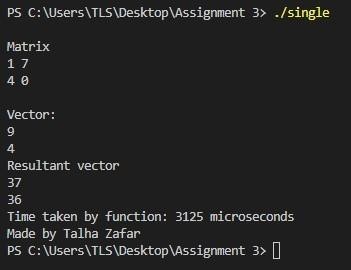
auto duration = duration\_cast<microseconds>(stop - start);

cout << "Time taken by function: " << duration.count() << " microseconds" <<

endl;

return 0;

Output



Multithreading: Code:

#include<stdio.h>

#include<pthread.h>

#include<unistd.h>

#include<stdlib.h>

#include <algorithm>

#include <chrono>

#include <iostream>

#include<vector>

using namespace std;

using namespace std::chrono;

#define MAX 2

#define mrow 2

#define mcol 2

#define vrow 2

#define vcol 1

void\* mult(void\* arg)

{

int\* data = (int\*)arg;

int k = 0, i = 0;

int x = data[0];

for (i = 1; i <= x; i++)

k += data[i] \* data[i + x];

int\* p = (int\*)malloc(sizeof(int));

\*p = k;

pthread\_exit(p);

}

int main()

{

auto start = high\_resolution\_clock::now();

int mat[mrow][mcol];

int vec[vrow][vcol];

int r1 = mrow, c1 = mcol, r2 = vrow, c2 = vcol;

for (int i = 0; i < r1; i++) {

for (int j = 0; j < c1; j++) {

mat[i][j] = rand() % 10;

}

}

for (int i = 0; i < r2; i++) {

for (int j = 0; j < c2; j++) {

vec[i][j] = rand() % 10;

}

}

cout << "Matrix:" << endl;

for (int i = 0; i < r1; i++) {

for (int j = 0; j < c1; j++)

printf("%d ", mat[i][j]);

printf("\n");

}

cout << "Vector:" << endl;

for (int i = 0; i < r2; i++) {

for (int j = 0; j < c2; j++)

printf("%d ", vec[i][j]);

printf("\n");

}

int max = r1 \* c2;

pthread\_t\* threads;

threads = (pthread\_t\*)malloc(max \* sizeof(pthread\_t));

int count = 0;

int\* data = NULL;

for (int i = 0; i < r1; i++)

for (int j = 0; j < c2; j++)

{

data = (int\*)malloc((20) \* sizeof(int));

data[0] = c1;

for (int k = 0; k < c1; k++)

data[k + 1] = mat[i][k];

for (int k = 0; k < r2; k++)

data[k + c1 + 1] = vec[k][j];

pthread\_create(&threads[count++], NULL,

mult, (void\*)(data));

}

printf("RESULTANT MATRIX IS :- \n");

for (int i = 0; i < max; i++)

{

void\* k;

pthread\_join(threads[i], &k);

int\* p = (int\*)k;

printf("%d ", \*p);

if ((i + 1) % c2 == 0)

printf("\n");

}

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

cout << "Time taken by function: " << duration.count() << " microseconds" <<

endl;

OutputText

Description automatically generated

Time Comparison Between Two Approaches:

|  |  |
| --- | --- |
| **Single Thread** | **Multithread** |
| 3125 microseconds | 5223 microseconds |

**Comparison**:

Single threaded processes involve executing instructions in a single sequence, meaning only one command is processed at a time. On the other hand, multithreaded processes allow for the execution of multiple parts of a program simultaneously. While single-threaded processes are faster than multithreaded ones, as they have a shorter execution time.