**Operating Systems - Lab 08 Tasks**

Ibrahim Johar Farooqi

23K-0074

BAI-4A

26 March 2025

**Lab Questions:**

**Task 1 - a:**

**Declare three float arrays A, B and C each of size 1e7 (10000000) and perform the operation C = A + B for each element of A, B and C.**

**a) Write, compile, and run serial code.**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define N 10000000 //10 million

float A[N], B[N], C[N];

void initalise()

{

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

{

//random values btw 0 to 100

A[i] = (float)(rand() % 100);

B[i] = (float)(rand() % 100);

}

}

void serial\_addition()

{

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

{

C[i] = A[i] + B[i];

}

}

int main()

{

clock\_t start, end;

double time\_taken;

initalise(); //A & B initialised

start = clock(); //start time

serial\_addition(); //computation performed here

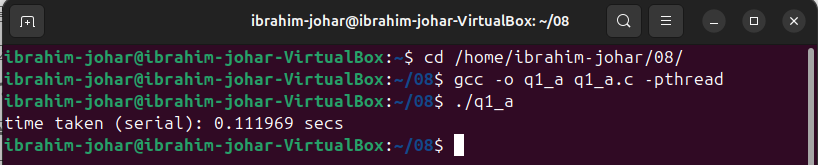
end = clock(); //end time

time\_taken = ((double)(end-start)) / CLOCKS\_PER\_SEC;

printf("time taken (serial): %.6f secs\n", time\_taken);

return 0;

}



**Task 1 - b:**

**b) Write concurrent code where 10 worker threads will equally divide the computational workload.**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <time.h>

#define N 10000000 //10 million

#define NUM\_THREADS 10 //num of worker threads

float A[N], B[N], C[N];

typedef struct {

int start;

int end;

} thread\_data;

void initalise()

{

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

{

//random values btw 0 to 100

A[i] = (float)(rand() % 100);

B[i] = (float)(rand() % 100);

}

}

void parallel\_addition(void\* arg)

{

thread\_data\* data = (thread\_data\*)arg;

for(int i = data->start; i < data->end; i++)

{

C[i] = A[i] + B[i];

}

pthread\_exit(NULL);

}

int main()

{

pthread\_t threads[NUM\_THREADS];

thread\_data th\_data[NUM\_THREADS];

clock\_t start, end;

double time\_taken;

initalise();

start = clock();

//creating threads

int chunk\_size = N / NUM\_THREADS;

for (int i = 0; i < NUM\_THREADS; i++)

{

th\_data[i].start = i \* chunk\_size;

th\_data[i].end = (i+1) \* chunk\_size;

pthread\_create(&threads[i], NULL, parallel\_addition, &th\_data[i]);

}

//joining threads

for (int i = 0; i < NUM\_THREADS; i++)

{

pthread\_join(threads[i], NULL);

}

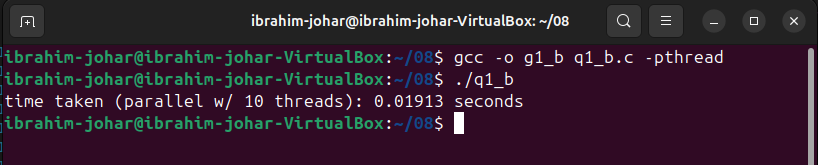
end = clock();

time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("time taken (parallel w/ %d threads): %.5f seconds\n", NUM\_THREADS, time\_taken);

return 0;

}



**Task 2:**

**Write a multithreaded program that calculates various statistical values for a list of numbers.This program**

**will pass a series of numbers on the command line and will then create three separate worker threads. One**

**thread will determine the average of the numbers, the second will determine the maximum value, and the**

**third will determine the minimum value. For example, suppose your program is passed the integers. (The array of numbers must be passed as parameter to threads, and the thread must return the calculated value to main thread).**

**90 81 78 95 79 72 85**

**The main thread will print:**

**The average value is 82.**

**The minimum value is 72.**

**The maximum value is 95.**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <limits.h>

typedef struct {

int \*numbers;

int count;

} thread\_data;

void\* calc\_avg(void\* arg) //thread func -> average

{

thread\_data\* data = (thread\_data\*)arg;

int sum = 0;

for (int i = 0; i < data->count; i++)

{

sum += data->numbers[i];

}

double\* avg = malloc(sizeof(double));

\*avg = (double)sum / data->count;

return avg;

}

void\* calc\_min(void\* arg) //thread func -> minimum

{

thread\_data\* data = (thread\_data\*)arg;

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

\*min = INT\_MAX; //initial large val

for (int i = 0; i < data->count; i++)

{

if (data->numbers[i] < \*min)

{

\*min = data->numbers[i];

}

}

return min;

}

void\* calc\_max(void\* arg) //thread func -> maximum

{

thread\_data\* data = (thread\_data\*)arg;

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

\*max = INT\_MIN; //initial small val

for (int i = 0; i < data->count; i++)

{

if (data->numbers[i] > \*max)

{

\*max = data->numbers[i];

}

}

return max;

}

int main(int argc, char\* argv[])

{

if (argc < 2)

{

printf("Usage: %s <list of numbers>\n", argv[0]);

return 1;

}

//converting cmd-line args to an int array

int count = argc - 1;

int nums[count];

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

{

nums[i] = atoi(argv[i+1]);

}

//data for threading

thread\_data data;

data.numbers = nums;

data.count = count;

pthread\_t threads[3];

//3 worker threads

pthread\_create(&threads[0], NULL, calc\_avg, &data);

pthread\_create(&threads[1], NULL, calc\_min, &data);

pthread\_create(&threads[2], NULL, calc\_max, &data);

//collect results

double\* avg;

int\* min;

int\* max;

pthread\_join(threads[0], (void\*\*)&avg);

pthread\_join(threads[1], (void\*\*)&min);

pthread\_join(threads[2], (void\*\*)&max);

printf("The average value is %.2f\n", \*avg);

printf("The minimum value is %d\n", \*min);

printf("The maximum value is %d\n", \*max);

//free allocated memory

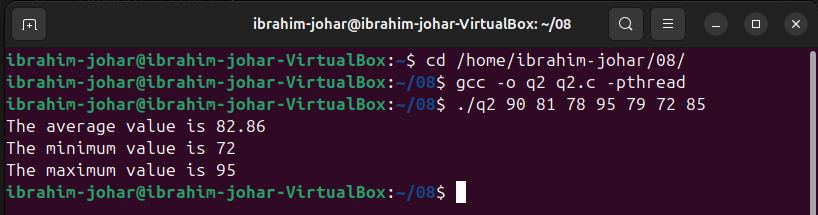
free(avg);

free(min);

free(max);

return 0;

}

****