//Anthony Zovich

//Dr. Joshi

//Program 3

//10/16/24

PROGRAM

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <math.h>

/\* the threads will set these values \*/

double average;

int maximum;

int minimum;

double median;

float standardDeviation=0.0;

//used to keep track of number of threads running

volatile int running\_threads = 0;

//number of threads to create

pthread\_t thread[5];

//length of array

int numOfElements;

//these methods are called when creating threads

/\*

//call these methods

void \*calculate\_average(void \*param);

void \*calculate\_maximum(void \*param);

void \*calculate\_minimum(void \*param);

\*/

void \*calculate\_average(void \*param)

{

//needs to be established outside of loop because we are using C

int i;

//get values of array

int \*values = (int\*)param;

//go through the array and add all values up

for(i = 0; i < numOfElements; i++){

average += values[i];

}

//divide by length of array

average = average/numOfElements;

//thread is over

running\_threads -= 1;

return NULL;

}

void \*calculate\_maximum(void \*param)

{

//copy code from average only find max

int i;

int \*values = (int\*)param;

//probably won't work with an array of negatives

maximum =0;

//go through array and find max

for(i = 0; i < numOfElements; i++){

if(values[i] > maximum){

maximum = values[i];

}

}

running\_threads -= 1; /\*Decrement thread count\*/

return NULL;

}

void \*calculate\_minimum(void \*param)

{

//repeat what was done with average only get min

int i;

int \*values = (int\*)param;

//minimum equals first value in array

minimum = values[0];

//if value at i is less than the previous minimum

//then assign minimum to the new smallest value

for(i = 0; i < numOfElements; i++){

if(values[i] < minimum){

minimum = values[i];

}

}

//thread is over

running\_threads -= 1;

return NULL;

}

//used to sort the array to find the median

int cmpfunc(const void \* a, const void \* b)

{

return(\*(int\*)a - \*(int\*)b );

}

void \*calculate\_median(void \*param)

{

//copy calculate min but instead find median

int i;

int \*values = (int\*)param;

//sort the array

qsort(values, numOfElements, sizeof(int), cmpfunc);

if(numOfElements%2==0) {

// if there is an even number of elements, return mean of the two elements in the middle

median = ((values[numOfElements/2] + values[numOfElements/2 - 1]) / 2.0);

} else {

// else return the element in the middle

median= values[numOfElements/2];

}

//thread is over

running\_threads -= 1;

return NULL;

}

void \*calculate\_standardDeviation(void \*param)

{

//use the mean(average) to calculate the standard Deviation

double mean;

int \*values = (int\*)param;

int i;

//sort the array

for(i = 0; i < numOfElements; i++){

mean += values[i];

}

//divide by length of array

mean = mean/numOfElements;

for(i =0; i<numOfElements; i++){

standardDeviation += pow((((float)values[i])- mean), 2);

//printf("%f", standardDeviation);

}

standardDeviation= (float) sqrt((standardDeviation/numOfElements));

//thread is over

running\_threads -= 1;

return NULL;

}

//taken from networks 317 to get number of elements and array

int getArrayInput(int n, int \*param){

int input;

//used to get length of array

int numberOfElements = 0;

//used from Networking project to take in an array In C

//infinite for loop because we can have up to infinite numbers

for(;;){

printf("Enter a positive value (A negative number will stop it: ");

//Get Int from console and throw it into the array

if (scanf("%d",&input) != 1){

printf("Need an integer");

exit(EXIT\_FAILURE);

}

if (input >= 0){

if (numberOfElements == n){

n += 1;

param = realloc(param, n \* sizeof(int));//realloc array and set pointer

}

//store input after last element

param[numberOfElements++] = input;

} else {

printf("\nNumber of Integers: %d \n", numberOfElements);

break;

}

}

//return the length of the array

return numberOfElements;

}

//join the 3 threads together

void joinThreads(int numberOfThreads){

int i; /\*count\*/

int s; /\*error #\*/

//join the threads together

while(numberOfThreads >= 0){

s = pthread\_join(thread[numberOfThreads], NULL);

//if s is not 0 then an error occured

if (s != 0){

//if it fails then exit and I'll fix later

exit(1);

}

numberOfThreads--;

}

}

//taken from networks 317 and edited for this project

void createThreads(int \*param){

int s;

//first thread will find the minimum

s = pthread\_create(&thread[0], NULL, calculate\_minimum, (void \*)param);

if (s != 0){

exit(1);

}

running\_threads += 1;

//create another thread to get maximum

s = pthread\_create(&thread[1], NULL, calculate\_maximum, (void \*)param);

if (s != 0){

exit(1);

}

running\_threads += 1;

//third thread will find average

s = pthread\_create(&thread[2], NULL, calculate\_average, (void \*)param);

if (s != 0){

exit(1);

}

running\_threads += 1;

//forth thread will find median

s = pthread\_create(&thread[3], NULL, calculate\_median, (void \*)param);

if (s != 0){

exit(1);

}

//fifth thread will find standard Deviation

s = pthread\_create(&thread[4], NULL, calculate\_standardDeviation, (void \*)param);

if (s != 0){

exit(1);

}

running\_threads += 1;

}

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

{

int n = 1; /\* Initial Array Size\*/

int \*param = malloc(n \* sizeof(int));/\*Initialize array pointer\*/

//get the inputed array from the user

numOfElements = getArrayInput(n, param);

//create the 3 threads

createThreads(param);

//wait for all of the threads to execute

while(running\_threads>0){

sleep(1);

}

//join the threads

joinThreads(4);

//print the results

printf("The average is %f\n", average);

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

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

printf("The median is %f\n", median);

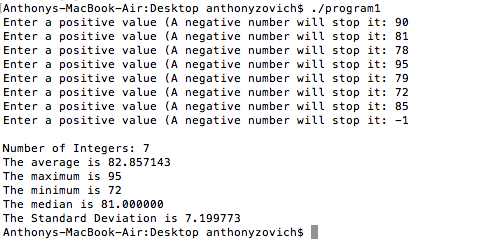
printf("The Standard Deviation is %f\n", standardDeviation);

return 0;

}

OUTPUT

Test 1



Test 2

