```
/*
This code demonstrates multithreading with consumer and producer threads to
calculate simple statistics on an array.
*/


#include <iostream>
#include <pthread.h>
#include <random>
#include <cmath>
#include <semaphore.h>


using namespace std;

//Here we delcare two semaphores one for when the buffer of the semaphore is empty
and full
//to define when the semaphore should lock and unlock
//A pthread mutex is also declared for thread locking/unlocking
sem_t e, full;
pthread_mutex_t m;
//We now define the number of threads as the number of resources and the array size
we will work with
#define num_resources 10
#define ARRAY_SIZE 1000

//p and j are count variables that are updated as the threads progress through the
program
int p = 0;
int j = 0;
//An array struct and constructor for the data type is declared
struct Array{
    int x;

    Array(int x_value = 0) : x(x_value){}
};


//Here we have the producer function where threads 1 - 10 enter the critical
sections
//to produce a unique array
void* producer(void* param){
    //generate 10 random arrays through the use of semaphores and pthreads

    //We seed the random number generator with the time and the thread count to get
a unique generation
    srand((unsigned int)time(NULL)+ p);

    //We point to the Array structure delcare a variable of its type
    //and allocate memory for an array in the heap per thread
    Array* random_array = new Array[ARRAY_SIZE];

    //the semaphore loads two processes and decrements to 0 putting the other
processes on stanby
    sem_wait(&e);
    //One thread out of two threads are loaded. The critical section is locked by
the pthreads mutex lock and the other thread is
    //spinning on stanby
    pthread_mutex_lock(&m);
    cout << endl << endl << "Producer Task started : Creating an Array : Thread # "
<< p + 1 << endl;
    for(int i = 0; i < ARRAY_SIZE; i++){

        //critical section
        //a random number between 0 - 250k is generated and placed at a position in
the array
        random_array[i].x = (int) rand() % 250000;

        }
    cout << "Producer Task finished creation of Array : Thread #" << p + 1 << endl
<< endl;
        //exiting critical section
    //The current thread operating on the critical section is released and another
thread is loaded
    //The thread is released and the spinning thread on stanby loads and completes
    pthread_mutex_unlock(&m);
    //The semaphore increments from 0 to 2 allowing the program to load two more
processes
    sem_post(&full);
    //After one pass through the count is incremented
    ++p;

    //As the thread exits it returns the produced array
    pthread_exit((void*)random_array);
}


void* consumer(void* param){
    //describe the statistics of 10 random arrays

    //We point to the array passed through the paramteres of pthread_creation as an
arguement of this function
    Array* random_array = (Array*)param;
    //Min and Max values are set to the index of the first position of the array to
comapre against
    //all other array values in the for loop.
    int min = random_array[0].x;
    int max = random_array[0].x;
    double average = 0.0;
    double standardDeviation = 0.0;
    long long sum = 0;

    sem_wait(&full);
    pthread_mutex_lock(&m);
    cout << "Consumer Task started : Creating Descriptive Statistics of an Array :
Thread #" << j + 1 << endl << endl;

        for(int a = 1; a < ARRAY_SIZE; a++){

            //cout << "number : " << a + 1 << "  : "<< random_array[a].x << endl;
            //Here we compare the value in the first position of the array index
against all other values
            //using a greater than and less than operand
            if(min > random_array[a].x){
                min = random_array[a].x;
            }
            if(max < random_array[a].x){
                max = random_array[a].x;
            }
            //We take the sum of all values within the array
            //To avoid an overflow we set the data type of sum as long long
            sum += random_array[a].x;

        }
        //To handle different data types of long long and int we use a static cast
to get
        //accurate computations
        average = static_cast<double>(sum) / static_cast<double>(ARRAY_SIZE);
        for(int a = 0; a < ARRAY_SIZE; a++){
            //To computer the standard deviation we compute sqrt(sum(x - mean)^2)
            //Step 1
            standardDeviation += pow(random_array[a].x - average,2);

        }
    //Step 2
    standardDeviation = sqrt(standardDeviation/ARRAY_SIZE);
    cout << "array length : " << ARRAY_SIZE << endl;
    cout << "the max number is: " << max << endl;
    cout << "the min number is: " << min << endl;
    cout << "the average is : " << average << endl;
    cout << "the standard deviation is : " << standardDeviation << endl;

    cout << endl << "Consumer Task finished for Thread #" << j + 1 << endl << endl;
    pthread_mutex_unlock(&m);
    sem_post(&e);
    j++;
    //We now delete the array we have allocated to memory to avoid duplicate arrays
    delete[] random_array;
    pthread_exit(0);
}


int main(){
    //We initalize the semaphore to handle two processes. One to decrement from 2 to
0 to load two processes and lock the others
    //and one to indicate that the semaphore is empty when its count is at 2
    sem_init(&e, 0, 2);
    sem_init(&full, 0, 0);
    //We initialize the mutex for thread locking in the critical section
    pthread_mutex_init(&m,NULL);

    //Two threads of an array type are created and initalized to 10
    //Indicating they will pass through their respective function 10 times
    pthread_t tid1[num_resources], tid2[num_resources];
    pthread_attr_t attr;

    pthread_attr_init(&attr);


    for(int i = 0; i < num_resources; i++){
        //create threads
        pthread_create(&tid1[i],&attr,producer,NULL);
        void* random_array;
        pthread_join(tid1[i], &random_array);
        pthread_create(&tid2[i],&attr,consumer,random_array);
        pthread_join(tid2[i], NULL);

    }

    return 0;

}
```