```
#include <iostream>
#include <time.h>
#include <sys/time.h>
#include <stdio.h>
#include <stdlib.h>
#include <cmath>
#include "TLS_Lib.h"
using namespace std;

/*Este programa compara el rendimiento de la especulación de lazos vs. la iteración no especulativa.
Para ello se implementa la Criba de Eratostenes, evaluando si 100 números son primos.

La versión especulativa consiste divide el problema en dos secciones, una primera especulación utilizada
para inicializar el vector, y una segunda que calcula los números primos. Para la extracción del paralelismo
se utilizó solo el lazo externo del método, asignándose cada iteración de este a una hebra distinta. Por este
motivo este programa ejemplo carece de escalabilidad: el sistema no podrá proveer un mayor número de hebras, de hecho
100 son bastantes hebras que pedir a un sistema común (por la forma del algoritmo, en realidad se le pedirá poco menos de la
mitad).

En una primera corrida el rendimiento del programa tradicional es similar al especulativo, esto es de
esperar puesto que el programa tradicional consiste exclusivamente de operaciones de lecturas y escrituras sobre la
data compartida. La versión especulativa realiza estas operaciones en un modo casi tan secuencial como el del programa
base, y además le añade el sobrecosto de la gestión del modelo, por todo esto las ventajas del paralelismo no son tan evidentes,
aunque por lo general el rendimiento de la especulación supera al tradicional.

Una evidente mejora del rendimiento se observaría si el lazo externo incluyese una carga adicional que no involucrase operaciones
estrictas sobre la data compartida. A fin de observar un caso semejante se realizó una segunda corrida del método anterior,
introduciendo una operación sleep(1) al final de cada iteración del lazo exterior. En el caso especulativo esta carga se solapa
entre las varias hebras, por lo cual se observará un rendimiento notablemente superior al caso secuencial.

 */

typedef struct {
        int             secs;
        int             usecs;

} TIME_DIFF;

TIME_DIFF * my_difftime (struct timeval * start, struct timeval * end)
{
        TIME_DIFF * diff = (TIME_DIFF *) malloc ( sizeof (TIME_DIFF) );

        if (start->tv_sec == end->tv_sec) {
                diff->secs = 0;
                diff->usecs = end->tv_usec - start->tv_usec;
        }
        else {
                diff->usecs = 1000000 - start->tv_usec;
                diff->secs = end->tv_sec - (start->tv_sec + 1);
                diff->usecs += end->tv_usec;
                if (diff->usecs >= 1000000) {
                        diff->usecs -= 1000000;
                        diff->secs += 1;
                }
        }

        return diff;
}

loop_speculator b, c;
int num_primes=100;

void* initialize (void* arg){
        int base=(((int)arg)-1)*(num_primes/10);
        int tope=((int)arg)*(num_primes/10);

        bool aux2=true;
        for (int i=base; i<tope; i++){
                bool* aux=((bool*&)b.get_shared_data()+i);
                while (b.write_data((bool*&)aux, (bool*)aux2)!=0){
                }
        }
        b.commit();
};

void* initialize_c (void* arg){
        int base=(((int)arg)-1)*(num_primes/10);
        int tope=((int)arg)*(num_primes/10);

        bool aux2=true;
        for (int i=base; i<tope; i++){
                bool* aux=((bool*&)c.get_shared_data()+i);
                c.write_data((bool*&)aux, (bool*)aux2);
        }
        c.commit();
};
```

```
void* get_primes (void* arg){
        bool aux2=false;
        for (int j=2; j<ceil((float)num_primes/(int)arg); j++){
                bool* aux=((bool*&)b.get_shared_data()+(j*(int)arg));
                b.write_data((bool*&)aux, (bool*)aux2);
        }
        b.commit();
}

void* get_primes_and_sleep (void* arg){
        bool aux2=false;
        for (int j=2; j<ceil((float)num_primes/(int)arg); j++){
                bool* aux=((bool*&)c.get_shared_data()+(j*(int)arg));
                c.write_data((bool*&)aux, (bool*)aux2);
        }
        c.commit();
        sleep(1);
}



int main()
{

        struct timeval myTVstart, myTVend;
        TIME_DIFF *     difference;


        bool vector_primos[num_primes];
        bool* argumento=&vector_primos[0];

        script_vector auxiliar;
        auxiliar.push_back(initialize); //Función a pasar para primera iteración
        vector <void*> constA;
        constA.push_back((void*)1); //Argumento de primera iteración.

        cout<<"Inicio del Programa"<<endl;

        //En primer lugar se inicializará el vector:

        cout<<"Inicio del cálculo especulativo."<<endl;

        gettimeofday (&myTVstart, NULL);

        b.speculate((void*&)argumento, auxiliar, constA); //Se manda a especular con solo la primera iteración.
        //Lo siguiente corresponderá al código pre-especulativo.

        for (int i=2; i<(num_primes/10)+1; i++){
                b.append(initialize, (void*)i); //Se añaden las demás iteraciones y argumentos.

        //Cada iteración inicializará 10 posiciones del vector.

        }
        b.commit(); //Se marca que la sección pre-especulativa ha terminado.
        b.get_results(); //Se piden los resultados de la especulación.

        //La segunda fase especulativa consiste en determinar los números que no son primos.
        //Lo siguiente corresponderá al código pre-especulativo.

        auxiliar.clear();
        auxiliar.push_back(get_primes); //Función a pasar para la primera iteración
        constA.clear();
        constA.push_back((void*)2); //Argumento de primera iteración.
        b.speculate((void*&)argumento, auxiliar, constA); //Se manda a especular con solo la primera iteración.

        for (int i=3; i<num_primes; i++){
                b.append(get_primes, (void*)i); //Se añaden las demás iteraciones y argumentos.
        }
        b.commit();//Se marca que la sección pre-especulativa ha terminado.
        b.get_results();  //Se piden los resultados de le especulación.

        gettimeofday (&myTVend, NULL);

        cout<<"Fin del cálculo especulativo."<<endl;

        cout<<"Primos calculados especulativamente: "<<endl;

        for (int i=2; i<num_primes; i++){
                if (vector_primos[i]){
                                cout<<" "<<i;
                }
        }
        cout<<endl;

        difference = my_difftime (&myTVstart, &myTVend);
```

```
printf ("Duración de la especulación (incluye wall-clock time): %d.%d segundos.\n", difference->secs, difference->usecs);

free (difference);


//Ahora se realiza el método no especulativo...

cout<<"Inicio del cálculo no especulativo."<<endl;

gettimeofday (&myTVstart, NULL);

for (int i=0; i<num_primes; i++){
        vector_primos[i]=true; //Inicialización.
}
for (int i=2; i<num_primes; i++){ //Dterminación de números no primos.
        for (int j=2; j<ceil((float)num_primes/i); j++){
                vector_primos[j*i]=false;
        }
}
gettimeofday (&myTVend, NULL);

cout<<"Fin del cálculo no especulativo."<<endl;

cout<<"Primos calculados no especulativamente: "<<endl;

for (int i=2; i<num_primes; i++){
        if (vector_primos[i]){
                        cout<<" "<<i;
        }
}
cout<<endl;

difference = my_difftime (&myTVstart, &myTVend);
printf ("Duración del método no especulativo (incluye wall-clock time): %d.%d segundos.\n", difference->secs, difference->usecs);

free (difference);


cout<<"Como es de esperar, el rendimiento del programa tradicional es casi similar al del programa especulativo (en algunos casos
cout<<"\n \nA continuación evaluaremos el mismo programa pero añadiendole una carga adicional a cada iteración del lazo externo"<<
cout<<"Esta carga representa operaciones que no involucran la data compartida, y que pueden ser cálculos temporales por iteración"
cout<<"del lazo externo. La carga se implementó como una invocación a sleep(1) al final de cada iteración del lazo externo.\n"<<en

auxiliar.clear();
constA.clear();

auxiliar.push_back(initialize_c); //Función a pasar para primera iteración
constA.push_back((void*)1); //Argumento de primera iteración.

cout<<"Segunda Corrida del Programa"<<endl;

//En primer lugar se inicializará el vector:

cout<<"Inicio del cálculo especulativo."<<endl;

gettimeofday (&myTVstart, NULL);

//Se usa otro especulador, para que el tiempo del reseteo del modelo no entre en el calculo.

c.speculate((void*&)argumento, auxiliar, constA); //Se manda a especular con solo la primera iteración.
//Lo siguiente corresponderá al código pre-especulativo.

for (int i=2; i<(num_primes/10)+1; i++){
        c.append(initialize_c, (void*)i); //Se añaden las demás iteraciones y argumentos.

//Cada iteración inicializará 10 posiciones del vector.

}
c.commit(); //Se marca que la sección pre-especulativa ha terminado.
c.get_results();  //Se piden los resultados de le especulación.

//La segunda fase especulativa consiste en determinar los números que no son primos.
//Lo siguiente corresponderá al código pre-especulativo.

auxiliar.clear();
auxiliar.push_back(get_primes_and_sleep); //Función a pasar para la primera iteración
constA.clear();
constA.push_back((void*)2); //Argumento de primera iteración.
cout<<"Inicio segunda especulaciòn..."<<endl;
c.speculate((void*&)argumento, auxiliar, constA); //Se manda a especular con solo la primera iteración.
for (int i=3; i<num_primes; i++){
        c.append(get_primes_and_sleep, (void*)i); //Se añaden las demás iteraciones y argumentos.
}
c.commit();//Se marca que la sección pre-especulativa ha terminado.
c.get_results();  //Se piden los resultados de le especulación.
```

```
        gettimeofday (&myTVend, NULL);

        cout<<"Fin del cálculo especulativo."<<endl;

        cout<<"Primos calculados especulativamente: "<<endl;

        for (int i=2; i<num_primes; i++){
                if (vector_primos[i]){
                                cout<<" "<<i;
                }
        }
        cout<<endl;

        difference = my_difftime (&myTVstart, &myTVend);
        printf ("Duración de la especulación (incluye wall-clock time): %d.%d segundos.\n", difference->secs, difference->usecs);

        free (difference);

        cout<<"Ahora se realizará el método no especulativo, lo que puede llevar un par de minutos"<<endl;

        //Ahora se realiza el método no especulativo...

        cout<<"Inicio del cálculo no especulativo"<<endl;

        gettimeofday (&myTVstart, NULL);

        for (int i=0; i<num_primes; i++){
                vector_primos[i]=true; //Inicialización.
        }
        for (int i=2; i<num_primes; i++){ //Determinación de números no primos.
                for (int j=2; j<ceil((float)num_primes/i); j++){
                        vector_primos[j*i]=false;
                }
                sleep(1);
        }
        gettimeofday (&myTVend, NULL);

        cout<<"Fin del cálculo no especulativo"<<endl;

        cout<<"Primos calculados no especulativamente: "<<endl;

        for (int i=2; i<num_primes; i++){
                if (vector_primos[i]){
                                cout<<" "<<i;
                }
        }
        cout<<endl;

        difference = my_difftime (&myTVstart, &myTVend);
        printf ("Duración del método no especulativo (incluye wall-clock time): %d,%d segundos.\n", difference->secs, difference->usecs);

        free (difference);

        cout<<"La ventaja más notoria de la especulación, aún bajo las limitaciones de esta implementación, proviene de permitir el solapa


}
```