UNCUYO UNIVERSIDAD NACIONAL DE CUYO | FACULTAD DE INGENIERÍA

Licenciatura en Ciencias de la Computación

Arquitecturas Distribuidas
Trabajo Práctico Nº1
Año 2023
Tema: Paralelismo a nivel de hilos

Introducción

En este trabajo práctico se pide resolver problemas paralelizables empleando la técnica de multihilos para disminuir el tiempo de ejecución o incrementar el throughput. Los problemas a resolver en este trabajo práctico también serán resueltos en el trabajo práctico N°3 “Paralelismo a nivel de procesos mediante MPI” para comparar ambas técnicas.

En todos los ejercicios se pide obtener datos sobre tiempos de ejecución para calcular speedups y visualizar porcentajes de uso de cada núcleo de su computadora. En los anexos al final de esta guía se provee información de cómo obtener esos datos, como también un resumen de las funciones necesarias, un repaso de C++ y un repaso de aritmética (puede ser necesario para algunos ejercicios).

No en todos los problemas obtendrá el mismo speedup, y deberá analizar el motivo.

Objetivos

- Resolver diferentes problemas con cierto grado de paralelismo mediante la técnica de multihilos empleando librerías y C++.
- Analizar las ventajas y limitaciones de esta técnica.

Metodología

Trabajo individual o grupal. 2 estudiantes por grupo máximo.

Tiempo de realización aproximado: 8 - 10 horas (4 - 5 clases).

Aprobación

- Mostrar en clases el buen funcionamiento de los programas escritos.
- Elaborar un informe indicando las características de la computadora en la cual ejecute los programas (cantidad de núcleos e hilos, puede usar comandos como lshw o lscpu) y para cada ejercicio el speedup logrado (comparando el tiempo utilizando un solo hilo y usando n hilos).
- Cargar en la plataforma Moodle:
    - El informe.
    - Los códigos fuentes de los programas escritos (no cargar ejecutables ni los archivos auxiliares del ejercicio 2).

Materiales necesarios

- Compilador de C++ versión C++11 o superior (Puede emplear un compilador en su máquina local o plataformas online como Repl.it o máquina virtual de Google Cloud). Las computadoras del laboratorio A de la facultad de Ingeniería están preparadas para compilar y ejecutar los programas solicitados.
- Opcional: IDE o herramienta de desarrollo de su preferencia.

Ejercicio Nº1:

Escribir un programa que calcule el logaritmo natural de números mayores a 1500000 ($1.5*10^6$) en punto flotante de doble precisión largo (tipo **long double** en C++) mediante serie de Taylor, empleando 10000000 (diez millones) de términos de dicha serie. El resultado debe imprimirse con 15 dígitos (ver anexo). Resuelva el problema de dos formas:

- Sin emplear multihilos.
- Empleando múltiples hilos que trabajen concurrentemente, resolviendo cada hilo una parte de los 10 millones de términos de la serie de Taylor (sugerencia, utilizar como número de hilos divisores de 10000000).

Tanto el operando del logaritmo natural como la cantidad de hilos a emplear deben ingresarse por teclado.

a) Incluya código que permita obtener el tiempo de ejecución en cada programa, y calcule el speedup (ver Anexo 1.4).

b) Observe el porcentaje de uso de cada núcleo en cada implementación (ver Anexo 1.6).

Serie de Taylor del logaritmo natural:

$$\ln(x) = 2\sum_{n=0}^{\infty} \frac{1}{2n+1}\left(\frac{x-1}{x+1}\right)^{2n+1}$$

Ayuda: ln(1500000)=14.2209756660724

Nota: Se sugiere no mezclar variables de diferentes tipos en una sola operación matemática. Algunas variables podrían desbordarse, y algunas arquitecturas podrían arrojar resultados según la variable de menor longitud. Puede usar máscaras para transformar todas las variables al formato requerido.

Ejercicio Nº2:

Se da un archivo de datos en forma de caracteres llamado “texto.txt” de 200 MB (200 millones de caracteres) , que puede descargar desde: https://drive.google.com/file/d/19Bnah1DCzZu5wJvFeuppcx3voUOtOEFS/view?usp=sharing, y 32 patrones, que constan en cadenas de caracteres almacenados una por línea en el archivo “patrones.txt” (que se encuentran en la carpeta del trabajo práctico N°1). Cree un programa que busque la cantidad de veces que cada patrón aparece en el archivo “texto.txt”. El programa deberá generar una salida similar a la

siguiente:

el patron 0 aparece 14 veces
el patron 1 aparece 3 veces
el patron 2 aparece 0 veces
el patron 3 aparece 0 veces
el patron 4 aparece 0 veces
el patron 5 aparece 0 veces
……..
……..

El archivo “texto.txt” no posee saltos de línea (es decir, posee solo una línea).
Resuelva el problema de dos formas:

- Sin emplear multihilos.
- Mediante 32 hilos que trabajen concurrentemente, de modo que cada hilo busque un patrón.

a) Incluya código que permita obtener el tiempo de ejecución en cada programa, y calcule el speedup (ver Anexo 1.4).
b) Observe el porcentaje de uso de cada núcleo en cada implementación (ver Anexo 1.6).

Nota: La búsqueda de patrones es un problema de gran importancia en seguridad informática.

Ayuda: El patrón 0 aparece 14 veces, el patrón 1 aparece 3 veces, el patrón 6 aparece 4 veces, el patrón 9 aparece 3622 veces, el patrón 11 aparece 2 veces, el patrón 13 aparece 6 veces, el patrón 16 aparece 2 veces, el patrón 18 aparece 6 veces, el patrón 21 aparece 2 veces, el patrón 27 aparece 6 veces, todos los demás patrones aparecen 0 veces.

Ejercicio N°3:
Escriba un programa que realice la multiplicación de dos matrices de N*N elementos del tipo **float** (número en punto flotante), y luego realice la sumatoria de todos los elementos de la matriz resultante. Muestre por pantalla los elementos de cada esquina de las matrices y el resultado de la sumatoria. El programa deberá permitir cambiar el número de elementos de las matrices (es decir, que el número de elementos sea N*N, donde N pueda cambiarse).
Resuelva el problema de dos formas:

- Sin emplear multihilos.
- Empleando múltiples hilos que trabajen concurrentemente, resolviendo cada hilo un grupo de las N filas de la matriz resultado (nota, usar N hilos no es la forma más eficiente de resolver el problema, 10 o 20 hilos son un número adecuado).

El programa deberá poder resolver productos de matrices de 3000*3000 elementos.
a) Incluya código que permita obtener el tiempo de ejecución en cada programa, y calcule el speedup (ver Anexo 1.4).
b) Observe el porcentaje de uso de cada núcleo en cada implementación (ver Anexo 1.6).

Ayuda:
Si los elementos de la matriz1 son 0.1, y los elementos de la matriz2 son 0.2, suponiendo matrices de 1000*1000 elementos, la matriz resultante será (se muestran solo los elementos en los extremos):

|20.0003 …… 20.0003|
| ……..|
|20.0003 …… 20.0003|

el resultado de la sumatoria será $2*10^7$.

Si el número de elementos de las matrices es 300*300, la matriz resultante será (se muestran solo los elementos en los extremos):

|6.000 …… 6.000|
| ……|
|6.000 …… 6.000|

el resultado de la sumatoria será 540000.

Si el número de elementos de las matrices es 3000*3000, la matriz resultante será (se muestran solo los elementos en los extremos):

|60.0012 …… 60.0012|
| ……|
|60.0012 …… 60.0012|

el resultado de la sumatoria será $5.71526*10^8$.

Nota: En la unidad 2 se estudiará la forma de resolver este problema de forma eficiente mediante diagramas de dependencia.

Ejercicio N°4:
Escriba un programa que busque todos los números primos menores a un número N que se ingresará por teclado. Debe mostrar por pantalla los 10 mayores números primos y la cantidad de números primos menores que N. Utilice como datos números del tipo ***long long int***.
Resuelva el problema de dos formas:

- Sin emplear multihilos.
- Empleando múltiples hilos que trabajen concurrentemente.

El valor de N deberá ser de al menos 10^7.
a) Incluya código que permita obtener el tiempo de ejecución en cada programa, y calcule el speedup (ver Anexo 1.4).
b) Observe el porcentaje de uso de cada núcleo en cada implementación (ver Anexo 1.6).
Ayuda:
Hay 78498 números primos menores que 10^6 (un millón), siendo los 5 mayores: 999953, 999959, 999961, 999979, 999983.
Hay 664579 números primos menores que 10^7 (diez millones), siendo los 5 mayores: 9999937, 9999943, 9999971, 9999973, 9999991.
Hay 5761455 números primos menores que 10^8 (diez millones), siendo los 5

mayores: 99999931, 99999941, 99999959, 99999971, 99999989.

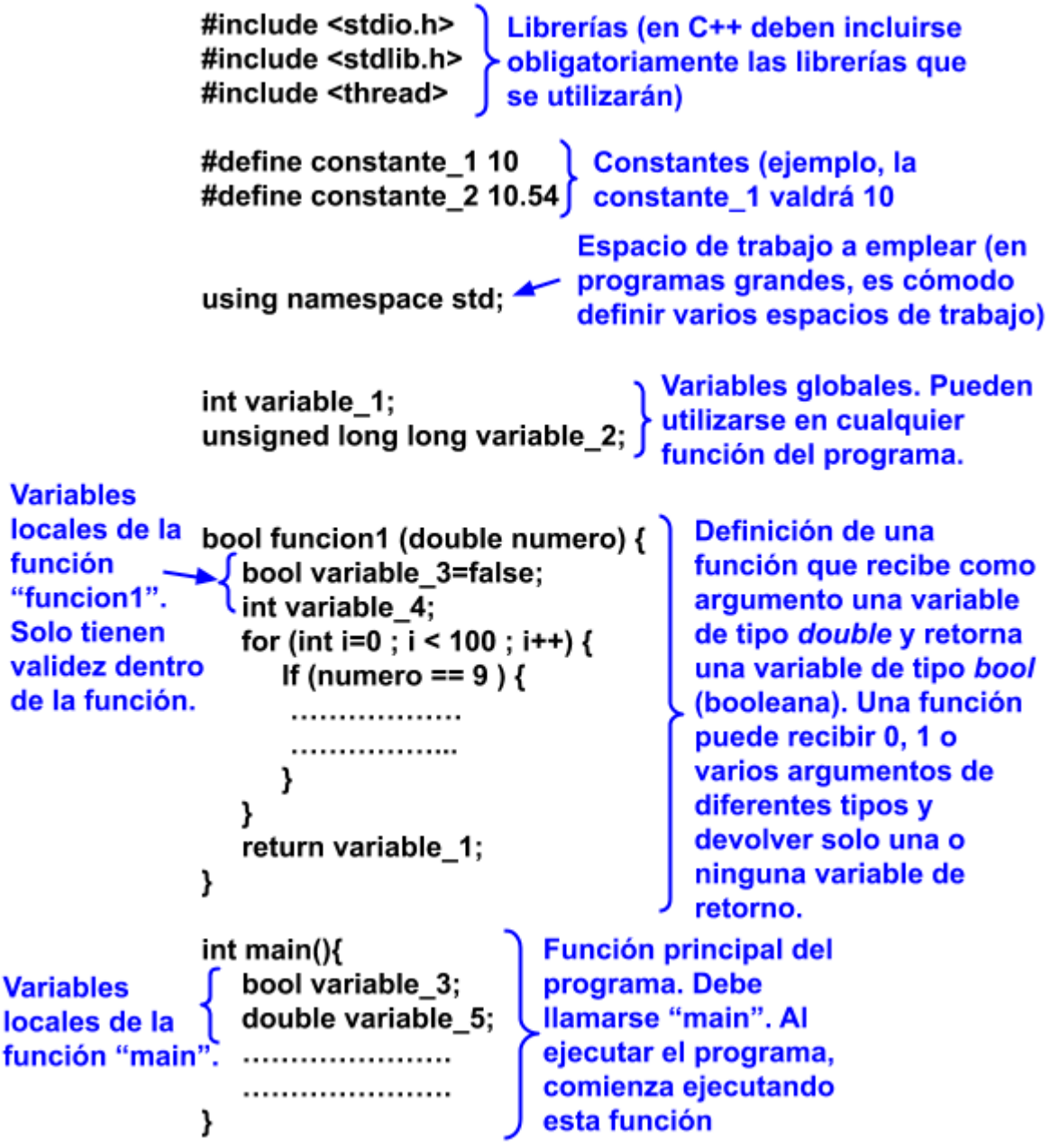
.

Nota 1: Resolver este problema requiere conocimientos de aritmética, uso de vectores de C++ y mecanismos de exclusión mutua. En el anexo 3 se brinda información para resolverlo.

Nota 2: Este problema requiere un poder de cómputo muy grande si los números son mayores a 10^9 o 10^{10} Con (C++ puede utilizar variables con tamaños hasta 1.1×10^{4932}), siendo el tipo de problemas que requiere grandes clusters para ser resuelto. La dificultad para resolver este problema es la base del algoritmo de encriptación RSA, el más seguro conocido hasta hoy día.

Anexo 1: Funciones útiles en C++

1.1 Estructura típica de un programa en C++



Recuerde que C++ es un lenguaje orientado a objetos, por lo que puede aplicar todos los conceptos sobre programación orientada a objetos vista en su carrera. Por ejemplo, deberá declarar objetos de clases predefinidas, teniendo estas clases diferentes constructores que podrá usar según necesite. Las páginas web www.cplusplus.com y https://www.w3schools.com/cpp/default.asp son las páginas web de consulta más habitual sobre este lenguaje. Para consultarla, deberá tener

claros los conceptos de herencia, constructores, destructores y miembros públicos de una clase.

1.2 Tipos de datos

Tipo de dato	representa	longitud	rango
char	caracteres	1 byte	Se utiliza para representar caracteres
int	enteros	2 bytes	-32767 a 32767
long int	enteros	4 bytes	-2147483648 a 2147483647
long long int	enteros	8 bytes	Máximo: 223372036854775807
float	punto flotante	4 bytes	-1.5×10^{-45} a 3.4×10^{38}
double	punto flotante	8 bytes	-5.0×10^{-345} a 1.7×10^{308}
long double	punto flotante	10 bytes	-3.4×10^{-4932} a 1.1×10^{4932}

Para consultar información sobre más tipos de datos, consultar: https://cplusplus.com/doc/tutorial/variables/

Los datos pueden variar para distintas arquitecturas. Puede usar la librería <climits> para conocer los valores máximos y mínimos de cada tipo de dato. Por ejemplo, las contantes ULLONG_MAX y ULLONG_MIN indican los números máximos y mínimos que la variable **unsigned long long** int puede almacenar.

unsigned: dato numérico sin signo: Agregando la palabra ***unsigned*** a cualquiera de los tipos de datos anteriores, la variable se convierte en una variable sin signo. Por ejemplo, la siguiente instrucción declara una variable de tipo entero de 2 bytes cuyo rango será de 0 a 65534:

unsigned int nombre_variable;

string: Variable tipo cadena de caracteres. Este tipo de variables es en realidad un tipo de objeto con varios métodos, se muestra uno de ellos. Declaración:

string buffer;

buffer.size(): indica la cantidad de caracteres de buffer:

Indicar la cantidad de dígitos con la cual se mostrará un número:
cout<< **setprecision(15)** <<numero<<endl;
Debe agregar la librería **#include <iomanip>**

Arrays
Debe indicar el tipo de variable del array y la cantidad de elementos. Puede tener

todas las dimensiones que se necesiten. Por ejemplo:

int matriz_1[100][100];

Otra forma de declarar un array es con:

int *matriz_resultados=new int[100];

El operador new crea un array de objetos del tipo indicado (en el ejemplo, del tipo **int**) y devuelve un puntero al primer objeto del array. Puede crear arrays de cualquier tipo de objetos, incluso de alguna clase definida por el usuario.

Vectores

Similar a List en Python. Permite declarar una colección de elementos sin necesidad de declarar el tamaño. Todos los elementos deben ser del mismo tipo. Debe incluir la librería **#include <vector>**. Declaración:

vector<int> nombre_del_vector;

Para conocer el número de elementos del vector:

nombre_del_vector.size();

Para acceder al elemento ***i*** del vector:

nombre_del_vector.at(i);

Agregar un elemento (por ejemplo, un número más) al final del vector:

nombre_del_vector.push_back(elemento);

También puede declarar vectores de vectores con.

vector<vector<int>>mi_matriz:

Puede encontrar otras funciones de la clase vectores en:
https://cplusplus.com/reference/vector/vector/

Potencia

(a^b, pudiendo a y b ser enteros o decimales). Debe incluir la librería ***math.h***. Declaración para a^b:

pow(a,b);

1.3 Manejo de archivos y cadenas de caracteres

Debe incluir la librería ***fstream***.

Abrir un archivo en modo solo lectura:

ifstream mi_archivo("nombre_archivo.txt");

O también:

ifstream mi_archivo;
mi_archivo.open("nombre_archivo.txt");

Note que en el primer caso, se crea un objeto empleando uno de los constructores de la clase ***ifstream*** que recibe como argumento el nombre del archivo. En el segundo caso, primero se crea el objeto, y luego se invoca su método ***open()***.

Leer una línea del archivo (hay varios métodos, abajo se describe uno):

getline(mi_archivo,buffer);

Donde ***buffer*** debe ser una variable de tipo ***string*** (cadena de caracteres).

Leer una cantidad N de caracteres

mi_archivo.read(buffer,N);

Puede leerse N caracteres, o llegar al final del archivo antes de leer los N caracteres. Si se llega al final del archivo, la variable **mi_archivo.eof()** toma el valor **true**.

Mostrar caracteres por pantalla (debe incluir la librería ***iostream***)

cout<< "ejemplo de texto a mostrar" << variable1 << buffer.size() << endl;

Donde ***endl*** es el caracter fin y salto de línea.

Buscar una cadena dentro de otra

string1.find (char* string2, posicion_inicial);

Busca la cadena string2 dentro de la cadena string1 desde la posición *posicion_inicial*. Las cadenas pueden ser declaradas como string o vector de char. Devuelve como resultado la posición donde string2 se encuentra dentro de string1. Si no encuentra string2 dentro de string1, devuelve ***string::npos***.
Esta función solo se puede aplicar sobre variables tipo string. Para aplicarla sobre cadena de caracteres el tipo array de char, debe aplicar una máscara, tal como *string(texto)*, suponiendo que texto es una variable del tipo array de char.

1.4 Medir tiempos de ejecución

Existen varios métodos. A continuación se muestra uno. Este método mide el tiempo entre dos ejecuciones de la instrucción ***gettimeofday***. Debe incluir la librería ***sys/time.h***:

```
timeval time1,time2;
gettimeofday(&time1,NULL);
 ……
 (Código cuyo tiempo de ejecución se desea medir)
 ……
gettimeofday(&time2,NULL);
cout << "Tiempo ejecución: " << double(time2.tv_sec - time1.tv_sec) +
+ double(time2.tv_usec-time1.tv_usec)/1000000 << endl;
```

(Las últimas dos líneas son una misma instrucción).

1.5 Compilar y ejecutar un programa con *g++*

Se recomienda fuertemente utilizar el compilador ***g++***. El mismo viene por defecto instalado en Linux. Debe compilar el programa para cada plataforma en se utilizará.
Puede escribir su programa en cualquier editor, por ejemplo, el editor de texto de Linux (gedit, el cual reconoce lenguaje C++). Para ello, cree un archivo con extensión ***.cpp*** y escriba su código en dicho archivo. También puede utilizar y probar la lógica de su código con plataformas online como Repl.it, pero para analizar la

performance y speedup de su código, deberá descargarlo y compilarlo en la máquina en la cual lo ejecutará, ya que el speedup depende del hardware de la computadora en la cual se ejecuta un programa.
Para compilar su programa, ejecutar:

g++ -std=c++11 -pthread -O3 -o nombre_ejecutable.out codigo_fuente.cpp

Donde:
g++ es el nombre del compilador.
-std=c++11 indica la versión de C++ a emplear (para este práctico, la versión 11 es suficiente y recomendada)
-pthread Indica al compilador incluir librerías de procesamiento de hilos. Si su programa no utiliza hilos, no es necesario incluir esta línea.
-O3 nivel de optimización. Existen diferentes niveles (O0, O1, etc.). Cada nivel implica mayor tiempo de compilación, pero ejecutables con menor tiempo de ejecución.
-o nombre_ejecutable.out nombre del archivo de salida ejecutable.
codigo_fuente.cpp nombre del archivo con el código fuente.

Para ejecutar un programa ya compilado, ejecute en una terminal:
./nombre_ejecutable.out

1.6 Escribir programas con código en varios archivos

Cuando el código de un programa se vuelve demasiado extenso, es común dividirlo en varios archivos. Por ejemplo, puede haber un archivo con la declaración de funciones, otro con las clases, etc.
C++ compila los archivos .cpp por separado, y luego ensambla los códigos objetos. Por lo tanto, en cada archivo de código, deben agregarse las librerías necesarias e indicar el espacio de trabajo.
Luego, en el archivo de código fuente principal, debe indicarse la ruta a los archivos de código con la directiva **include<ruta/archivo.cpp>**, o **include "archivo.cpp"** si el archivo está en la misma carpeta que el archivo principal.

1.7 Visualizar porcentaje de uso de cada núcleo y memoria

Opción 1: top (Solo Linux)
En una terminal, ejecute la aplicación ***top***, presione ***1*** para ver el porcentaje de uso de cada núcleo, y luego presione "**t**" para ver el porcentaje de uso de forma gráfica. Presione "**m**" para ver el porcentaje de uso de memoria.

Opción 2: Monitor del sistema de Linux
Utilice la aplicación "Monitor del sistema", empleando la opción "Recursos"

Anexo 2: Manejo de hilos con C++

Debe incluir la librería ***thread*** e incluir el flag ***-pthread*** cuando compile.

Crear un hilo. Debe incluir el código a ejecutar por el hilo en una función que no devuelva ningún valor de retorno (tipo ***void***), y reciba todos los argumentos que sean necesarios. Luego crear y trabajar con el hilo.
Ejemplo de función:

```
void nombre_funcion(int arg1, long arg2){
        ……..
        Código de la función que implementa el hilo
        ……
}
```

Luego, para crear y ejecutar el hilo, puede utilizar dos opciones (en este práctico, será más útil la segunda opción).

1° opción:

```
thread nombre_hilo(nombre_funcion, argumento1, argumento2, ….);
```

Note que se está utilizando el constructor de la clase ***thread***.

2° opción:

```
thread nombre_hilo;
nombre_hilo=thread(nombre_funcion, argumento1, argumento2, ….);
```

Note que primero se crea una variable de tipo ***thread***, y luego a dicha variable se le asigna un objeto creado con el constructor de la clase ***thread***.

Sincronización de hilos mediante **join**:
Existen muchos métodos para sincronizar hilos. En este práctico será de gran utilidad (siempre necesario) el método ***join()***. Este método es útil cuando un hilo debe esperar a que otro hilo finalice su ejecución. Se utiliza como sigue:

```
if(hilo_1.joinable()==true){
        hilo_1.join();
}
```

El hilo en el cual se ejecute la instrucción ***join()*** se bloqueará hasta que el ***hilo_1*** finalice su ejecución. Note que no se puede ejecutar ***hilo_1.join()*** si ***hilo_1*** ya ha finalizado su ejecución, por eso primero se debe verificar si ***hilo_1.joinable()*** es igual a ***true***.
Note que ejecutar ***hilo_1.join()*** dentro del código del ***hilo_1*** producirá que el programa se bloquee.

Sincronización mediante **mutex**
Otro método para sincronizar hilos es **mutex**. Mutex es útil cuando se desea que solo un hilo pueda acceder a un recurso a la vez. Por ejemplo, dos hilos agregando

o quitando elementos de vectores puede conducir a errores.
C++ implementa mutex mediante objetos. Se debe declarar primero un objeto de la clase ***mutex***, y luego invocar los métodos ***lock()*** y ***unlock()*** (entre otros)***.*** Ejemplo:

```
mutex espera_escritura;
……..
espera_escritura.lock();
……
espera_escritura.unlock();
```

El primer hilo que ejecute ***espera_escritura.lock()*** bloqueará a los demás hilos que lleguen a dicha instrucción (note que es una carrera), hasta que el hilo bloqueante ejecute ***espera_escritura.unlock();*** Este mecanismo puede conducir a deadlock si no se utiliza con cuidado.

Arrays de hilos:
Cuando se usan hilos que trabajan en paralelo para resolver un problema, es muy útil declarar arrays de hilos. Ejemplo:

```
thread array_de_hilos[numero_hilos];
for(int i=0;i<numero_hilos;i++){
        array_de_hilos[i] = thread(nombre_funcion, arg1, arg2);
}
for(int i=0 ; i<numero_hilos ; i++){
        if(hilos[i].joinable()==true){
                hilos[i].join();
        }
}
```

Note que no puede bloquear el hilo principal del programa con ***join()*** hasta haber creado todos los hilos, de lo contrario, creará y ejecutará un solo hilo a la vez, y no habrá paralelismo.

Anexo 3: Repaso de aritmética de los números primos

Un número primo es aquel que solo es divisible por 1 y por si mismo. Todo número puede factorizarse en números primos (a este enunciado se lo llama teorema fundamental del álgebra). Por ejemplo, los factores de 20 son 2, 2 y 5, ya que 20=2*2*5. Por lo tanto, 20 no es un número primo.
0 y 1 no son considerados primos.

Dificultades para resolver el ejercicio 4:
No existe un algoritmo simple y rápido para detectar si un número es primo. La única forma forma de saber si un número K es primo, es dividirlo por ***todos los números primos*** menores o iguales a $\sqrt{K}$, y verificar que el resto de la división sea diferente de cero para todas las divisiones.
La dificultad de hallar todos los números primos menores que N mediante paralelismo, es que cada hilo (o proceso) debe ir buscando números primos y

escribirlos en una lista o vector de números primos que irá creciendo, pero esa lista lista será usada por cada uno de los hilos para buscar otros números primos. Esto puede conducir a que un hilo trabaje con datos incompletos. Tampoco puede imponer que un hilo no comience su tarea hasta que otro finalice, porque no estaría aprovechando el paralelismo.
Encontrar un algoritmo eficiente y confiable que resuelva este problema es un problema abierto hoy día.

Un algoritmo simple para hallar los números primos:
Para hallar todos los números primos entre 0 y N, propondremos un algoritmo simple que sigue tres pasos:
1) Busca los números primos entre 0 y $N^{1/2}$. Almacenarlos en un vector auxiliar.
2) Creamos y ejecutamos los hilos que trabajarán en paralelo. Cada hilo busca números primos en una parte del intervalo de números naturales entre 2 y N probando si los números impares “i” son divisibles o no por alguno de los *números primos* entre 2 y $N^{1/2}$ previamente encontrados (notar que los números pares no son primos).
3) Ordenamos el vector resultante. Como los hilos buscan números primos de forma concurrente, los números escritos en el vector de números primos probablemente no estén ordenados.
Nota: este algoritmo es ineficiente por varias razones, entre estas: Los hilos que trabajen con números mayores tendrán mayor carga de trabajo.