



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: TISTA GARCÍA EDGAR

Asignatura: ESTRUCTURA DE DATOS Y ALGORITMOS II

Grupo: 5

No de Práctica(s): PRÁCTICA # 11 – INTRODUCCIÓN A OPENMP

Integrante(s): CARRILLO CERVANTES IVETTE ALEJANDRA

*No. de Equipo de
cómputo empleado:* TRABAJO EN CASA

No. de Lista o Brigada: 09

Semestre: 2022 - 1

Fecha de entrega: 27 NOVIEMBRE 2021

Observaciones:

CALIFICACIÓN: _____

PRÁCTICA #11 – Introducción a OpenMP

Objetivo: El estudiante conocerá y aprenderá a utilizar algunas de las directivas de OpenMP utilizadas para realizar programas paralelos.

Esta práctica se realizó en la distribución Ubuntu del sistema operativo Linux, con la finalidad de probar cada uno de los códigos que contiene la guía de estudios impartida por la Facultad, estos códigos serán en el lenguaje de programación C; mientras que, los ejercicios proporcionados por el profesor se realizaron en Visual Studio Code, estos códigos serán en el lenguaje de programación JAVA.

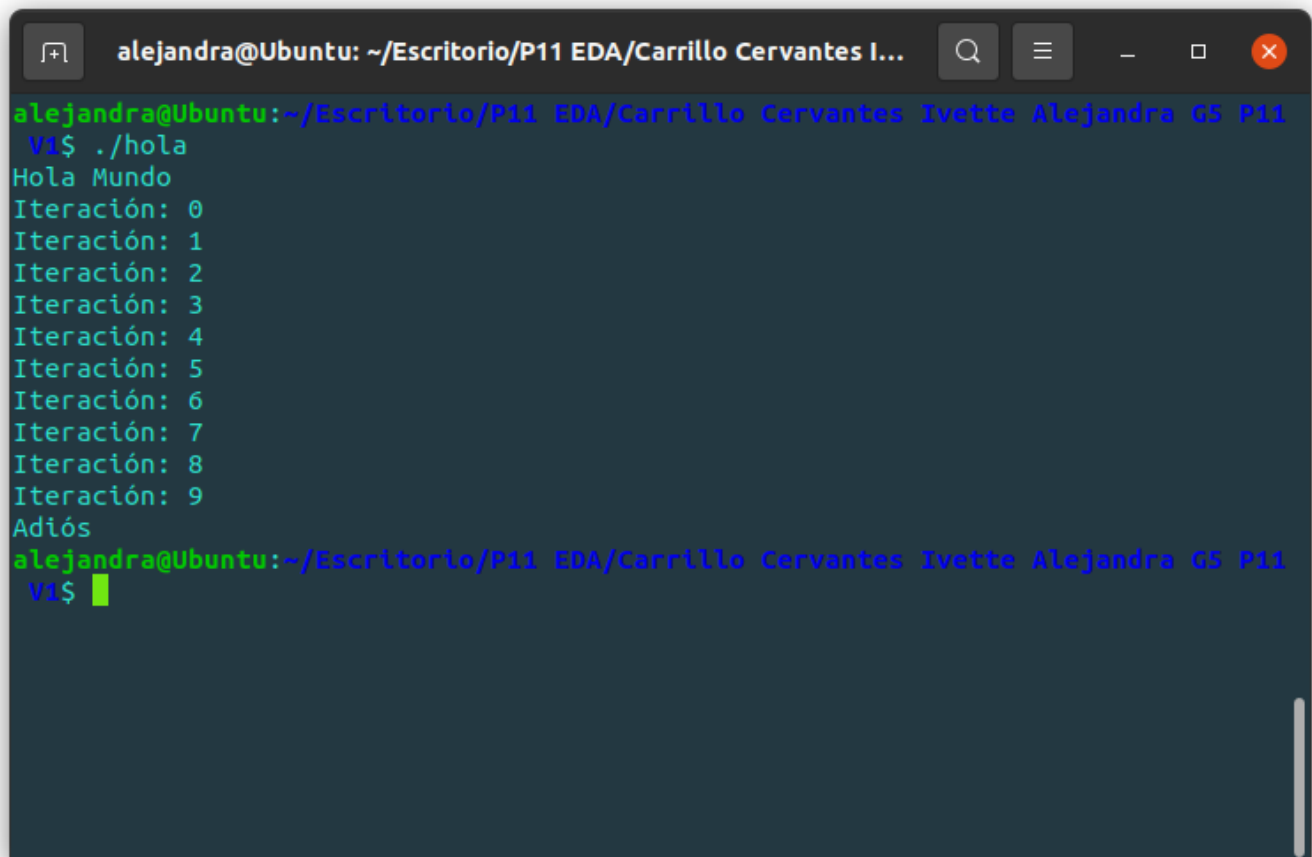
ACTIVIDADES PARA EL DESARROLLO DE LA PRACTICA

Sección 1 Ejercicios de la guía (OPEN MP)

Código Original

El código Original con el que se trabajará la mayoría de las actividades que se encuentran en la guía de estudios impartida por la facultad consiste en escribir un “Hola Mundo”, seguido de un ciclo for que se repetirá 10 veces e irá imprimiendo cada una de las iteraciones de este. Este programa se ejecutó de manera normal en C: “`gcc hola.c -o hola`”

La salida del código sin modificaciones es la siguiente:



```
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra G5 P11
Vi$ ./hola
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Adiós
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra G5 P11
Vi$
```

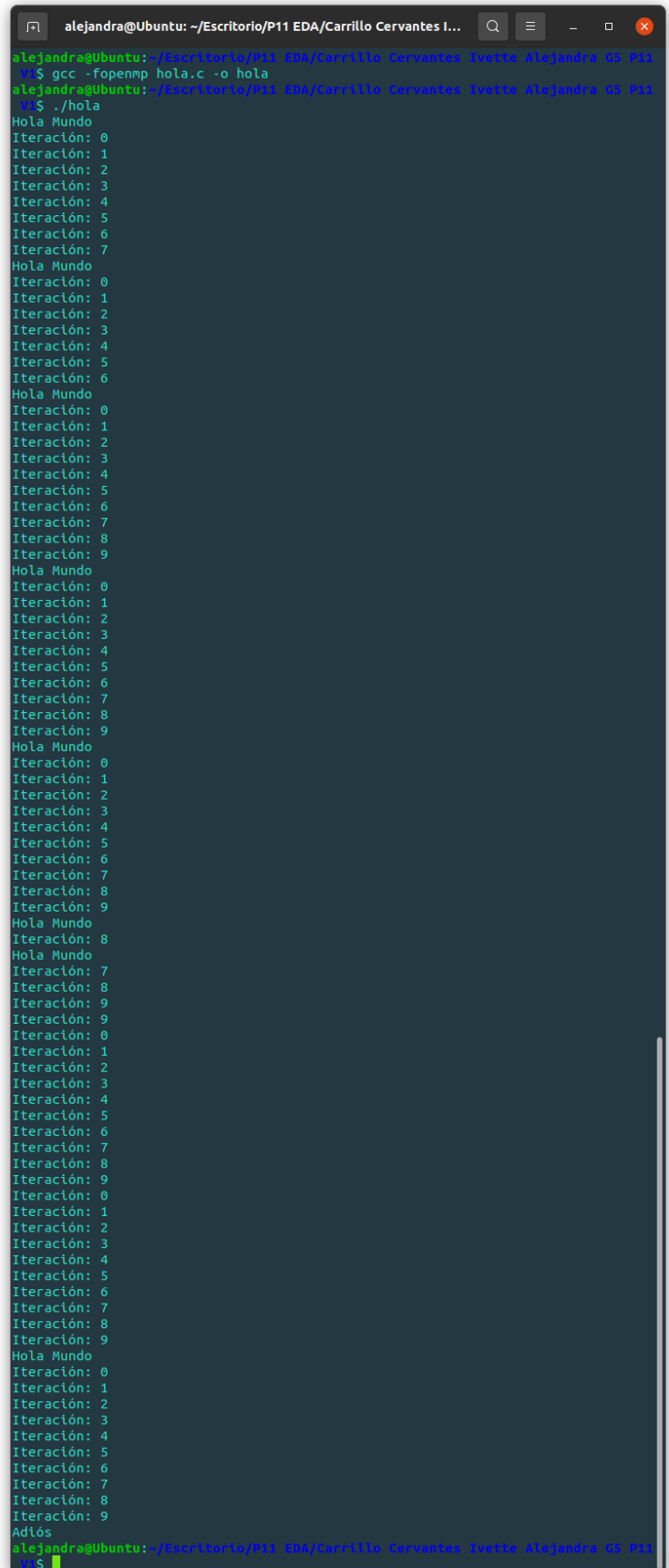
Código Original

NOTA: Este código se irá modificando con forme se realice cada actividad.

Actividad 1

Una vez ejecutado el código original, se formó una región paralela desde la declaración de la variable hasta antes de la última impresión a pantalla que es la impresión de “adiós”, esto se logró con ayuda del constructor **parallel**, agregando la bandera en compilación “**-fopenmp**” indicando que se agregarán las directivas de openMP.

Al momento de ejecutarlo podemos observar que la principal diferencia entre este y el primero, es que la primera ejecución en el código original no se trabajaba con hilos ni de una forma paralela y se puede ejecutar de una manera sencillo trabajando con un solo hilo y una sola forma de que se vaya repitiendo el ciclo for; mientras que, en esta segunda ejecución al formar una región paralela se formará un grupo de hilos los cuales querrán ir ejecutando el programa todos y la salida no tendrá un orden en específico, puesto que al querer irse ejecutado todos al mismo tiempo se va encimado unos de otros.

A terminal window titled 'alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes I...' showing the execution of a program. The user runs 'gcc -fopenmp hola.c -o hola' and then './hola'. The output consists of multiple interleaved lines of 'Hola Mundo' and 'Iteración: 0' through 'Iteración: 9', demonstrating non-deterministic execution order due to parallel processing. The terminal ends with 'Adiós' and a prompt for the next command.

```
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes I...
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes I...$ gcc -fopenmp hola.c -o hola
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes I...$ ./hola
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Hola Mundo
Iteración: 8
Hola Mundo
Iteración: 7
Iteración: 8
Iteración: 9
Iteración: 9
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Adiós
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes I...$
```

Forma paralela

Actividad 2

Dado el ejercicio anterior, se modificó el código, con el fin de que en vez de que se generen varios hilos aleatorios e la región paralela, se puedan generar un número de hilos predefinidos previamente, para ello se tuvo que modificar la variable de ambiente `"OMP_NUM_THREADS"` desde la consola, indicando la cantidad de hilos que se quiera crear, incluyendo la biblioteca `"omp.h"` y agregado la cláusula `"num_threads(n)"` seguida después del constructor `"parallel"`. Ahora la salida de este programa será el número de iteraciones, o bien ciclos de repetición for, correspondientes a el número de hilos que se le indicó al programa.

A continuación, se observa dos ejemplos, al modificar el número de hilos a 4 y 2 respectivamente.

```

alejandra@Ubuntu: ~/Escritorio/P11 EDA/...
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette
Alejandra GS P11
V1$ export OMP_NUM_THREADS=4
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette
Alejandra GS P11 V1$ gcc -fopenmp hola.c -o hola
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette
Alejandra GS P11 V1$ ./hola
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Adiós

```

Ejecución del programa con 4 hilos

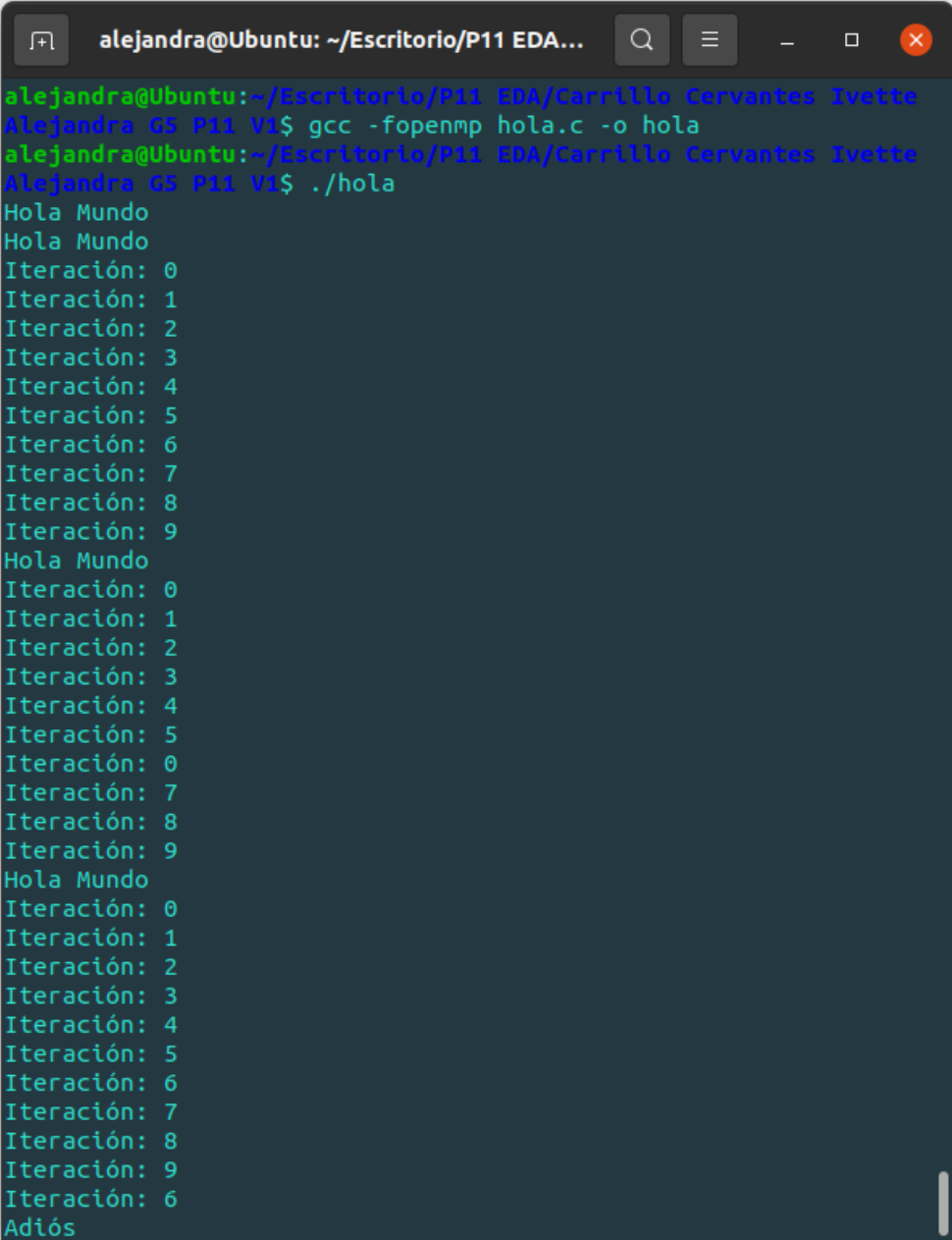
```
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes Ivette
Alejandra G5
P11 V1$ export OMP_NUM_THREADS=2
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette
Alejandra G5 P11 V1$ gcc -fopenmp hola.c -o hola
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette
Alejandra G5 P11 V1$ ./hola
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Adiós
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette
Alejandra G5 P11 V1$
```

Ejecución del programa con 2 hilos

Actividad 3

Dado el programa del ejercicio 1, se modificó la región paralela con el fin de que la variable que almacena las iteraciones del ciclo for quedará fuera de esta región, con el fin de observar la condición de carrera, la cual se da cuando en un programa varios hilos tienen acceso concurrente a una sola dirección de memoria, en el caso de este ejercicio la variable o bien, dirección de memoria será la variable "i" dado que se encuentra fuera de la región paralela y será compartida por los hilos correspondientes (recordemos que los hilos que se establecieron e u principio fueron 4).

Después de que este programa se compiló y ejecuto varias veces, se observó que todos los hilos trataban de acceder y cambiar el valor de la variable "i" que se encuentra fuera de la región.



```
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes Ivette
Alejandra G5 P11 V1$ gcc -fopenmp hola.c -o hola
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette
Alejandra G5 P11 V1$ ./hola
Hola Mundo
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 0
Iteración: 7
Iteración: 8
Iteración: 9
Hola Mundo
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Iteración: 6
Adiós
```

race condition

Actividad 4

Dado el código del ejercicio anterior, se modificó para que una variable compartida (fuera de la región) sea privada para que la ocupen los hilos creados independientemente, es decir cada hilo tendrá una variable privada "i" y cada uno pueda acceder sin necesidad de compartir dicha variable y sea propia de este hilo. Esta variable no tendrá ningún valor al final de la ejecución.

Cabe recalcar que, la cláusula "private" tiene lugar después del constructor parallel, el cual tiene como parámetro la variable que se desea hacer privada.

```
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra G5 P11
VI$ gcc -fopenmp hola4.c -o hola4
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra G5 P11
VI$ ./hola4
Hola Mundo
Hola Mundo
Iteración: 0
Iteración: 1
Hola Mundo
Iteración: 0
Iteración: 1
Hola Mundo
Iteración: 2
Iteración: 3
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Iteración: 0
Iteración: 1
Iteración: 0
Iteración: 1
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Iteración: 2
Iteración: 3
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Iteración: 4
Iteración: 5
Iteración: 6
Iteración: 7
Iteración: 8
Iteración: 9
Adiós
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra G5 P11
VI$
```

Cláusula private()

Actividad 5

En esta actividad se vio el uso del identificador de cada uno de los hilos mediante las funciones `omp_get_num_threads()` y `omp_get_thread_num()`, esto se logró indicando que la variable principal sea privada dentro de la región paralela con el fin de que cada hilo tuviera una variable “única” que pueda utilizar y no se obtuviera un “race control”. Se observó que al quitar la cláusula `private`, no tiene un orden cada uno de los hilos.

```
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra GS P11
VI$ gcc -fopenmp hola5.c -o hola5
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra GS P11
VI$ ./hola5
Hola Mundo desde el hilo 0 de un total de 4
Hola Mundo desde el hilo 3 de un total de 4
Hola Mundo desde el hilo 1 de un total de 4
Hola Mundo desde el hilo 2 de un total de 4
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra GS P11
VI$
```

Actividad 6_1

```
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra GS P11
VI$ gcc -fopenmp hola6_1.c -o hola6_1
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra GS P11
VI$ ./hola6_1
3      6      7      5      3      5      6      2      9      1
2      7      0      9      3      6      0      6      2      6
5      13     7      14     6      11     6      8      11     7
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra GS P11
VI$
```

Actividad 6_2

```
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra GS P11
VI$ gcc -fopenmp hola6_2.c -o hola6_2
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra GS P11
VI$ ./hola6_2
3      6      7      5      3      5      6      2      9      1
2      7      0      9      3      6      0      6      2      6
hilo 0 calculo C[0]= 5
hilo 0 calculo C[1]= 13
hilo 0 calculo C[2]= 7
hilo 0 calculo C[3]= 14
hilo 1 calculo C[5]= 11
hilo 1 calculo C[6]= 6
hilo 1 calculo C[7]= 8
hilo 1 calculo C[8]= 11
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra GS P11
VI$
```


Actividad 7

Para esta actividad se observó el uso del constructor for, el cual divide las iteraciones de una estructura de repetición for en una región paralela, este constructor detecta varios hilos que pueden trabajar con el mismo algoritmo o instrucciones que se repetirán para que cada hilo trabaje con su propia variable i.

```
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra G5 P11
VI$ gcc -fopenmp hola7.c -o hola7
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra G5 P11
VI$ ./hola7
Hola Mundo
Hola Mundo
Iteración:3
Iteración:4
Hola Mundo
Iteración:8
Iteración:9
Iteración:0
Iteración:1
Iteración:2
Hola Mundo
Iteración:6
Iteración:7
Iteración:5
Adiós
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra G5 P11
VI$
```

Actividad 8

```
alejandra@Ubuntu: ~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra G5 P11
VI$ gcc -fopenmp hola8.c -o hola8
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra G5 P11
VI$ ./hola8
3      6      7      5      3      5      6      2      9      1
2      7      0      9      3      6      0      6      2      6
hilo 0 calculo C[0]= 5
hilo 0 calculo C[1]= 13
hilo 0 calculo C[2]= 7
hilo 1 calculo C[3]= 14
hilo 1 calculo C[4]= 6
hilo 1 calculo C[5]= 11
hilo 2 calculo C[6]= 6
hilo 2 calculo C[7]= 8
hilo 3 calculo C[8]= 11
hilo 3 calculo C[9]= 7
alejandra@Ubuntu:~/Escritorio/P11 EDA/Carrillo Cervantes Ivette Alejandra G5 P11
VI$ ~
```

Sección 2. Semáforos en Java

Actividad 1

La clase semáforo es una clase publica la cual implementa el objeto Serializable. Al instanciar un objeto de esta clase, tiene como objetivo mantener un conjunto de permisos para controlar el uso de los hilos en Java.

Los métodos constructores de es esta clase son los siguientes:

- ➔ Semaphore(int permits). Tiene como parámetro el número de permisos que tendrá. los hilos.
- ➔ Semaphore(int permits, boolean fair). Tiene como parámetro el Número de permisos y la configuración de equidad dada.

Algunos de los métodos más importantes de esta clase, son los siguientes:

- ➔ Acquire(). Es un método sobrecargado, tiene como propósito adquirir un permiso de un semáforo bloqueándolo hasta que haya uno disponible o se interrumpa el hilo.
- ➔ drainPermits(). Adquiere y devuelve todos los permisos que están disponibles de inmediato o, si hay permisos negativos disponibles, los liberan.
- ➔ tryAcquire(). Es un método sobrecargado el cual, tiene como propósito adquirir un número de permisos del semáforo sólo si hay uno disponible en el momento de mandar a llamarlo.
- ➔ release(). Es un método sobrecargado, emite un permiso, devolviéndolo al semáforo.
- ➔ getQueueLength(). Devuelve una estimación del número de subprocesos que esperan adquirir.
- ➔ hasQueuedThreads(). Consulta si hay un subproceso a espera de adquirir.
- ➔ isFair(). Devuelve verdadero si este semáforo tiene la equidad establecida como verdadero.

Actividad 2 -> SemaphoreDemo

Este ejercicio tiene como función principal observar el funcionamiento de los hilos en el lenguaje de programación Java, el cual se logra a través de la herencia de la clase Thread. En su método principal se inicializan 2 hilos A y B, y posteriormente a ellos se empiezan mediante el método start(). En este ejercicio se sobrescribió el método run, el cual es necesario para empezar a ejecutar los hilos y se puso la condición de que si el primero que se ejecutaba era el hilo A entonces se procedía a sumar una variable de clase, sin embargo, si era el hilo B se debería de restar dicha variable, ambos casos se repetían 5 veces en un ciclo de repetición For, al igual que en cada uno de estos se dormía 10 milisegundos, con el fin de que el otro hilo terminará su ejecución. Todo esto se trató como una excepción coma ya que puede haber un error al momento de interrumpir la ejecución de un hilo. Cabe recalcar que en cada iteración sí imprimían el contador de la variable de clase y al final al sumar o restar estas variables debería de cero.

<pre> Starting A A is waiting for a permit. Starting B B is waiting for a permit. A gets a permit. A: 1 A: 2 A: 3 A: 4 A: 5 A releases the permit. B gets a permit. B: 4 B: 3 B: 2 B: 1 B: 0 B releases the permit. count: 0 </pre>	<pre> Starting A Starting B B is waiting for a permit. A is waiting for a permit. B gets a permit. B: -1 B: -2 B: -3 B: -4 B: -5 B releases the permit. A gets a permit. A: -4 A: -3 A: -2 A: -1 A: 0 A releases the permit. count: 0 </pre>
---	--

Actividad 2 -> SemaphoreTest

Esta actividad tiene la función de ir asignando permisos a cada uno de los hilos creados como en este caso se crearon hilos con nombre con nombre de la letra A la letra F y se les fue dando y quitando permisos a cada uno de estos.

```

Total available Semaphore permits : 4
A : acquiring lock...
F : acquiring lock...
C : acquiring lock...
C : available Semaphore permits now: 4
B : acquiring lock...
F : available Semaphore permits now: 4
D : acquiring lock...
E : acquiring lock...
A : available Semaphore permits now: 4
E : available Semaphore permits now: 2
D : available Semaphore permits now: 2
F : got the permit!
B : available Semaphore permits now: 3
C : got the permit!
F : is performing operation 1, available Semaphore permits : 0
E : got the permit!
A : got the permit!
E : is performing operation 1, available Semaphore permits : 0
C : is performing operation 1, available Semaphore permits : 0
A : is performing operation 1, available Semaphore permits : 0
F : is performing operation 2, available Semaphore permits : 0
C : is performing operation 2, available Semaphore permits : 0
E : is performing operation 2, available Semaphore permits : 0
A : is performing operation 2, available Semaphore permits : 0
F : is performing operation 3, available Semaphore permits : 0
E : is performing operation 3, available Semaphore permits : 0
C : is performing operation 3, available Semaphore permits : 0
A : is performing operation 3, available Semaphore permits : 0
C : is performing operation 4, available Semaphore permits : 0
E : is performing operation 5, available Semaphore permits : 0
C : is performing operation 5, available Semaphore permits : 0
A : is performing operation 5, available Semaphore permits : 0
F : is performing operation 5, available Semaphore permits : 0

```

```

C : releasing lock...
A : releasing lock...
A : available Semaphore permits now: 1
F : releasing lock...
E : releasing lock...
F : available Semaphore permits now: 1
B : got the permit!
B : is performing operation 1, available Semaphore permits : 2
D : got the permit!
C : available Semaphore permits now: 1
D : is performing operation 1, available Semaphore permits : 2
E : available Semaphore permits now: 2
D : is performing operation 2, available Semaphore permits : 2
B : is performing operation 2, available Semaphore permits : 2
D : is performing operation 3, available Semaphore permits : 2
B : is performing operation 3, available Semaphore permits : 2
B : is performing operation 4, available Semaphore permits : 2
D : is performing operation 4, available Semaphore permits : 2
D : is performing operation 5, available Semaphore permits : 2
B : is performing operation 5, available Semaphore permits : 2
D : releasing lock...
B : releasing lock...
B : available Semaphore permits now: 4
D : available Semaphore permits now: 3

```

Conclusiones

Se cumplió con el Objetivo de esta práctica, ya que se conocieron algunas directivas de Open MP utilizando el lenguaje de programación C con el fin de realizar programas paralelos; además, se vio el uso de hilos en el lenguaje de programación Java, los cuales se implementaron mediante la herencia de la clase Thread.

Al principio de esta práctica se me complico un poco entenderle a la ejecución tanto de los ejercicios de la guía, y de los que dio el profesor, ya que no comprendía del todo el tema; sin embargo, al investigar un poco más acerca de la programación paralela tanto en el lenguaje de programación C, como en Java comprendí un poco más acerca de este tema. Y a pesar de que no entendí muy bien el tema por completo, considero que aprendí lo básico para manejar este tema.

No se cumplió con todos los ejercicios de la práctica, ya que el ejercicio 6 y 8 de la guía no pude completarlo, al momento de describir su implementación, puesto que no entendí del todo bien al momento de pasar un hilo al otro; pero, considero que al practicar más programas implementando este tema, poco a poco irá quedándome más claro.

Considero que este es un tema de gran impacto al momento de realizar otros programas, ya que hay aplicaciones que se pueden utilizar simultáneamente y es necesario el uso de hilos, incluso antes tenía la duda de como realizar algunas aplicaciones en donde se pudieran ejecutar dos acciones, pero al estudiar este tema, me quedo un poco más claro.