



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

M.I. Edgar Tista García

Profesor:

Estructura de Datos y Algoritmos II

Asignatura:

5

Grupo:

11 "Introducción a OpenMP"

No de Práctica(s):

Calzada Martínez Jonathan Omar
García Lazcano Carlos David

Integrante(s):

*No. de Equipo de cómputo
empleado:*

34 y 35

No. de Lista o Brigada:

9 y 14

Semestre:

2019-2

Fecha de entrega:

7 de mayo de 2019

Observaciones:

CALIFICACIÓN: _____

Objetivos: El alumno conocerá y aprenderá a utilizar algunas Directivas de Open MP utilizadas para realizar programas paralelos

Introducción: La computación paralela es una forma de cómputo donde múltiples instrucciones se ejecutan al mismo tiempo ya sean en un mismo o como en distintos equipos. Esta forma de computación es comúnmente utilizada en arquitecturas de alto rendimiento como son los supercomputadores.

Desarrollo:

Ejercicio 1.

Lo que podemos observar a continuación se refiere al manejo de hilos, normalmente estamos acostumbrados, nosotros como alumnos, a que todos nuestros programas se realicen una vez, sin embargo con el concepto de hilos podemos ver como se puede realizar una tarea en cada hilo, de esta manera se podría decir que se ejecutan varias veces(todo depende de los núcleos con los que cuente nuestra máquina) .

Por ejemplo en el siguiente ejecutable se puede apreciar que se imprimen 8 veces casi la misma instrucción, pero con diferentes valores, sin embargo solo se ha hecho una vez y como el equipo empleado tiene 8 “núcleos” estos son los que realizan dicha actividad, también se puede observar que no tienen un orden en el cual digan que uno tiene que actuar después del otro esto quiere indicar que el manejo de los hilos son completamente manejados por el sistema operativo.

```
Hola Mundo desde el hilo 3 de un total de 8
Hola Mundo desde el hilo 5 de un total de 8
Hola Mundo desde el hilo 4 de un total de 8
Hola Mundo desde el hilo 1 de un total de 8
Hola Mundo desde el hilo 2 de un total de 8
Hola Mundo desde el hilo 7 de un total de 8
Hola Mundo desde el hilo 6 de un total de 8
Hola Mundo desde el hilo 0 de un total de 8
Adios
Process returned 0 (0x0)   execution time : 0.227 s
Press any key to continue.
```

Programa 1, guía de la práctica:

Hola mundo con y sin omp

Sin omp-

Con omp

```
D:\Users\Jonathan\Desktop\2019-2\EDAI\practica11\hola.c
Hola mundo
Iteracion: 0
Hola mundo
Iteracion: 1
Iteracion: 0
Hola mundo
Iteracion: 1
Hola mundo
Iteracion: 2
Iteracion: 0
Hola mundo
Iteracion: 3
Iteracion: 0
Hola mundo
Iteracion: 1
Iteracion: 0
Hola mundo
Iteracion: 2
Hola mundo
Iteracion: 3
Iteracion: 0
Iteracion: 2
Iteracion: 1
Iteracion: 4
Iteracion: 1
Iteracion: 2
Iteracion: 0
Iteracion: 2
Iteracion: 1
Iteracion: 3

Process returned 0 (0x0)   execution time :
Press any key to continue.
```

```
Iteracion: 4
Iteracion: 5
Iteracion: 6
Iteracion: 7
Iteracion: 8
Iteracion: 9
Adios
```

¿Qué diferencia hay con una y con otra?+

En la primera se puede observar que se hace una iteración del 0 al 9 de manera secuencial por medio de alguna operación repetitiva en este caso fue un for en donde se imprime la palabra “Iteración” y se imprime el número del índice hasta que esto haya pasado 10 veces y luego muestra la palabra “Adiós” y en la segunda se realiza lo mismo pero con omp se puede ver como al principio esta todo encimado y luego se acomoda.

¿Por qué son diferentes?

Son diferentes debido a que se está trabajando con los 8 núcleos y esto quiere decir que se estará ejecutando la misma ejecución 8 veces y cada núcleo estará imprimiendo en pantalla de manera en como valla ejecutándose por lo que pueden amontonarse en un principio.

Actividad 2). Cambiar el número de hilos a n

Haciendo las modificaciones correspondientes cambiando los hilos se mostró lo mismo pero la imagen de arriba pero solo con 4 núcleos por lo que se imprimió 4 veces.

En el siguiente se puede observar que usamos 3 tipos de suma, para ver más rápido y fácil cada manera implementada que nos sugiere la guía de la práctica, puesto que, en la parte inferior a donde dice “primero” hace una simple suma de los vectores, en los siguientes dos se usa el paralelismo, de manera que, los dos hilos en ejecución se dividen la tarea, y cada vez que acabé uno el otro sigue con su proceso y al terminar cede el procesador, de esta manera se puede decir que se realizan las labores más fácilmente, puesto que el problema es atendido por dos hilos, reduciendo así la labor si se hiciese por uno solo.

```

1      7      4      0      9      4      8      8      2      4
5      5      1      7      1      1      5      2      7      6

Primero
6      12      5      7      10      5      13      10      9      10
Parallel 1
hilo 0 calculo C[0] = 6
hilo 1 calculo C[5] = 5
hilo 0 calculo C[1] = 12
hilo 1 calculo C[6] = 13
hilo 0 calculo C[2] = 5
hilo 1 calculo C[7] = 10
hilo 0 calculo C[3] = 7
hilo 1 calculo C[8] = 9

Parallel 2
hilo 1 calculo C[5] = 5
hilo 0 calculo C[0] = 6
hilo 1 calculo C[6] = 13
hilo 0 calculo C[1] = 12
hilo 1 calculo C[7] = 10
hilo 0 calculo C[2] = 5
hilo 1 calculo C[8] = 9
hilo 0 calculo C[3] = 7
hilo 1 calculo C[9] = 10
hilo 0 calculo C[4] = 10

Process returned 0 (0x0)   execution time : 0.255 s
Press any key to continue.

```

Ejercicio 2

```

C:\Users\Carlo\Desktop\practica 11\ejercicio2>javac Prime.java

C:\Users\Carlo\Desktop\practica 11\ejercicio2>java Prime
no command line argument

C:\Users\Carlo\Desktop\practica 11\ejercicio2>java Prime abc
improper format

C:\Users\Carlo\Desktop\practica 11\ejercicio2>java Prime 0
command line argument 0 is too small

C:\Users\Carlo\Desktop\practica 11\ejercicio2>java Prime 10
printing primes from 2 to 10
2 is prime
3 is prime
5 is prime
7 is prime

```

También lo usamos para 100

```
C:\Users\Carlo\Desktop\practica 11\ejercicio2>java Prime 100
printing primes from 2 to 100
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
```

Ejercicio 2.

3. Starting Java threads

En el documento primero nos da un repaso de hilos en donde nos comienza explicando desde que es un proceso y nos dice que es un programa en ejecución y que a este se le ha asignado memoria administrada por el sistema operativo. En donde nos comenta que un Subproceso es una ejecución o flujo de control en el espacio de direcciones de un proceso.

Al mismo tiempo un proceso es un programa con al menos un hilo y nos dice que un proceso puede tener más de un hilo. Donde todos los subprocesos dentro de un proceso tienen su propio contador y su propia pila local y direcciones de retorno.

En el lenguaje java un hilo en el intérprete de tiempo de ejecución se llama método main ()

Los multiprocesadores de memoria compartida son más baratos y más comunes por esto cada hilo puede ser asignado a un CPU Este es menos costoso y más eficiente crear varios subprocesos en un proceso y puede compartir datos que crear varios procesos que comparten datos.

La I/O en dispositivos lentos (redes, terminales y discos) se puede hacer en un hilo mientras que otro hilo hace computo útil en paralelo y varios subprocesos pueden manejar los eventos (como clics en el mouse) en multiples ventanas en el Sistema.

En un clúster de estaciones de trabajo LAN o en un entorno de sistema operativo distribuido, el servidor que se ejecuta en una máquina puede generar un hilo para manejar una solicitud entrante el paralelo al hilo principal que continúa aceptando solicitudes entrantes adicionales.

Se explica que **cuando dos hilos realizan una función** tal como $N = N + 1$ en aproximadamente el mismo tiempo, tiene una condición de carrera. Ambos hilos son "de carreras" entre sí para acceder a los datos y una de las actualizaciones puede perderse.

Al mismo tiempo la ejecución de hilos que comparten datos necesitan sincronizar sus operaciones y tratamiento para evitar las condiciones de carrera sobre los datos compartidos. La sincronización de hilos se puede hacer con las variables de bandera y espera ocupada. Debido a que utiliza una gran cantidad de ciclos de CPU, espera ocupada es ineficiente. El bloqueo sería mejor.

Una **sección crítica** es un bloque de código en un subproceso que accede a una o más variables compartidas en forma de lectura-actualización-escritura. En tal situación, queremos una exclusión mutua en la que solo un hilo puede acceder (leer-actualizar-escribir) una variable compartida a la vez.

4. Thread states, priorities, and methods (Estados de hilo, prioridades y métodos Estados de hilo)

Los estados del hilo se definen como:

- * Nuevo antes de que se llame al método inicio del hilo()
 - * Ejecuta si el hilo está en la cola de espera
 - * Ejecuta si el hilo se está ejecutando en el CPU
 - * Muerto después de que se complete el método run () del hilo o se llame al método stop ()
 - * Bloqueado si el subproceso está bloqueado en E / S, una llamada al método join () o un método de suspensión (ms) llamada
 - * Suspendido si el método suspend () del hilo se llama desde run o en ejecución
- estados
- * Bloqueo suspendido si se llama al método suspend () del hilo desde el estado bloqueado
- Las prioridades de los hilos (variables de clase) son:

```
* MAX_PRIORITY
* NORM_PRIORITY
* MIN_PRIORITY
```

Cuanto mayor sea el número entero, mayor será la prioridad. En cualquier momento dado, cuando hay varios subprocesos listos para ejecutarse, el sistema de tiempo de ejecución elige el subproceso ejecutable con la prioridad más alta para la ejecución. Solo cuando ese hilo se detenga, ceda o no pueda ejecutarse por algún motivo, se iniciará un hilo de menor prioridad. Si dos subprocesos de la misma prioridad esperan a la CPU, el programador elige uno de ellos para que se ejecute de forma rotatoria.

Time slicing

También conocida como programación round-robin está hecho por la JVM esta clase no garantiza el tiempo de corte por lo que solo funciona en la práctica.

5. The volatile modifier

El **modificador volátil** le dice al compilador que se accede a la variable por más de una

hilo a la vez e inhibe las optimizaciones de código inapropiadas por el compilador, como cachear el valor de la variable en un registro de CPU en lugar de actualizar la memoria principal con cada asignación a la variable.

La especificación del lenguaje Java garantiza que las actualizaciones de cualquier variable compartida por un hilo particular es visto por otros hilos en el orden realizado por ese hilo en particular.

Sin embargo, el JLS no requiere otros subprocesos para ver las actualizaciones de diferentes variables compartidas

en el orden realizado por el hilo de actualización a menos que las variables se declaren volátiles.

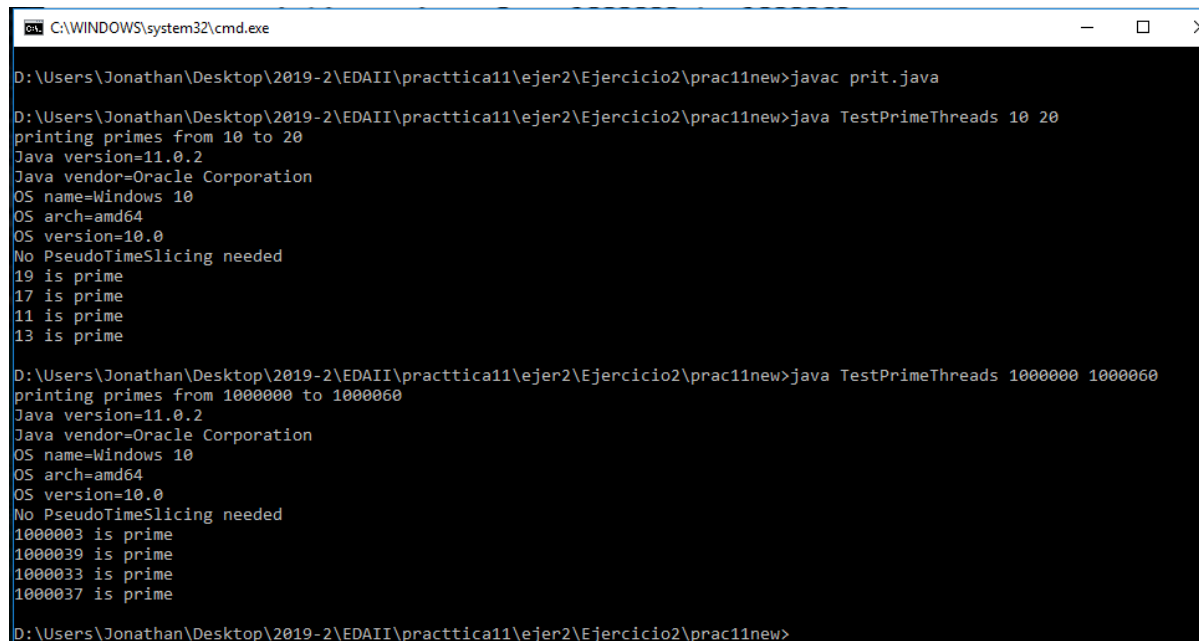
El ejemplo bwbb.java en la página 19 implementa un buffer acotado de espera para un productor y consumidor. El subproceso productor deposita elementos y espera ocupado si se llena el buffer acotado.

La hebra del consumidor obtiene elementos y espera ocupada si el búfer acotado está vacío.

El hilo productor debe "ver" el valor y los campos ocupados actualizados por el consumidor hilo en el orden exacto en que las actualizaciones son realizadas por el hilo del consumidor (o el consumidor debe ver las actualizaciones del productor en el orden realizado). Si no, el hilo productor podría sobrescribir un elemento en una ranura de almacenamiento intermedio que el consumidor aún no ha leído (o el consumidor podría volver a leer un elemento de una ranura de almacenamiento intermedio que ya ha leído). Driver bbdr.java en la página 20

Crea los hilos de productor y consumidor.

Compilando prit:



```
C:\WINDOWS\system32\cmd.exe
D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>javac prit.java
D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>java TestPrimeThreads 10 20
printing primes from 10 to 20
Java version=11.0.2
Java vendor=Oracle Corporation
OS name=Windows 10
OS arch=amd64
OS version=10.0
No PseudoTimeSlicing needed
19 is prime
17 is prime
11 is prime
13 is prime

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>java TestPrimeThreads 1000000 1000060
printing primes from 1000000 to 1000060
Java version=11.0.2
Java vendor=Oracle Corporation
OS name=Windows 10
OS arch=amd64
OS version=10.0
No PseudoTimeSlicing needed
1000003 is prime
1000039 is prime
1000033 is prime
1000037 is prime

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>
```

Compilado prime

Las instrucciones que recibe al compilar es estricto, no se puede pasar "Strings" debido a que necesita números enteros para saber ya que lo que regresa el programa es cuales de los números que están dentro del rango de 2 y el número proporcionado son números primos.

```

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>javac Prime.java

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>java prime
Error: Could not find or load main class prime
Caused by: java.lang.NoClassDefFoundError: Prime (wrong name: prime)

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>java Prime
no command line argument

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>java Prime abc
improper format

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>java Prime 0
command line argument 0 is too small

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>java Prime 10
printing primes from 2 to 10
2 is prime
3 is prime
5 is prime
7 is prime
D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>

```

Se puede decir que ordenamos que para realizar cierta cantidad de labores lo repartimos entre 4 núcleos

```

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>javac Beeper.java

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>java Beeping 4 100000 no 3
Beeping: numBeepers=4, beep=100000, timeSlice=no, runTime=3
Beeper0 is alive, beep=100000
Beeper1 is alive, beep=100000
Beeper2 is alive, beep=100000
Beeper3 is alive, beep=100000
All Beeper threads started
age()=72, Beeper0 running
age()=73, Beeper3 running
age()=73, Beeper1 running
age()=73, Beeper2 running
age()=102, Beeper3 beeps, value=100001
age()=106, Beeper2 beeps, value=100001
age()=102, Beeper0 beeps, value=100001
age()=106, Beeper1 beeps, value=100001
age()=128, Beeper3 beeps, value=200001
age()=130, Beeper2 beeps, value=200001

age()=3068, Beeper3 beeps, value=138600001
age()=3067, Beeper2 beeps, value=137200001
age()=3067, Beeper0 beeps, value=137600001
age()=3072, Beeper1 beeps, value=138700001
age()=3081, Beeper3 beeps, value=138700001
age()=3081, Beeper2 beeps, value=137300001
age()=3081, Beeper0 beeps, value=137700001
age()=3082, Beeper1 beeps, value=138800001
age()=3082, Beeper3 beeps, value=138800001
age=3073, time to interrupt the Beepers and exit
age()=3082, Beeper2 beeps, value=137400001
age()=3087, Beeper0 beeps, value=137800001
age()=3087, Beeper3 beeps, value=138900001
age()=3087, Beeper1 beeps, value=138900001
age=3086, Beeper2 interrupted
age=3098, Beeper1 interrupted
age=3098, Beeper3 interrupted
age=3098, Beeper0 interrupted
All Beeper threads interrupted

```

Compilado quad

En esta compilación se muestra la versión de java instalado, la corporación, el sistema operativo utilizado , el tipo de arquitectura y la versión del del sistema.


```

age=3098, Beeper0 interrupted
All Beeper threads interrupted

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>javac quad.java

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>java AdaptiveQuadrature 0.5 1.5 0.001
Adaptive Quadrature of x**2 from 0.5 to 1.5 with relative error 0.001
Java version=11.0.2
Java vendor=Oracle Corporation
OS name=Windows 10
OS arch=amd64
OS version=10.0
No PseudoTimeSlicing needed
Result for x**2 = 1.084136962890625

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>javac bwbb.java bldr.java

D:\Users\Jonathan\Desktop\2019-2\EDAI\practtica11\ejer2\Ejercicio2\prac11new>java ProducersConsumers 10 1 1 2 2 5
ProducersConsumers:
 numSlots=10, numProducers=1, numConsumers=1, pNap=2, cNap=2, runTime=5
Java version=11.0.2
Java vendor=Oracle Corporation
OS name=Windows 10
OS arch=amd64
OS version=10.0
No PseudoTimeSlicing needed
All threads started
age=82, Consumer0 napping for 493 ms
age=82, PRODUCER0 napping for 535 ms
age=600, Consumer0 wants to consume
age=641, PRODUCER0 produced item 0.9796286795315022
age=659, PRODUCER0 deposited item 0.9796286795315022
age=659, Consumer0 fetched item 0.9796286795315022
age=660, Consumer0 napping for 140 ms
age=660, PRODUCER0 napping for 1299 ms
age=802, Consumer0 wants to consume
age=1962, PRODUCER0 produced item 0.9819695208655255
age=1962, PRODUCER0 deposited item 0.9819695208655255
age=1963, PRODUCER0 napping for 684 ms
age=1962, Consumer0 fetched item 0.9819695208655255
age=1964, Consumer0 napping for 1336 ms

```

Conclusiones:

Calzada Martínez Jonathan Omar:

En general los objetivos de la práctica se cumplieron ya que se pudo comprender la manera en cómo se trabaja con la programación paralela y de este modo realizar la práctica con éxito, con ayuda claro de los archivos proporcionados y de la teoría vista en clase. Se aprendió a trabajar de manera práctica con la programación paralela por lo que esto nos ayudó bastante a enriquecer nuestros conocimientos. Se nos complicó un poco la parte de la traducción debido a que no sabemos inglés al 100% y se manejan algunos tecnicismos propios de la computación, así mismo también estábamos un poco confundidos de como correr/ ejecutar los programas propuestos en los archivos proporcionados por el profesor hasta nos dimos cuenta que todo era por consola.

García Lazcano Carlos David:

Los objetivos de la practica se cuimplieron puesto que a mipunto de vista fue más fácil comprender cómo funciona el paralelismo. Tener un punto de partida fue fundamental, puesto que al tratar de leer sin tener los conceptos dados en clases y las explicaciones se complicó tanto la ejecución como la comprensión, si bien estábamos acostumbrados o dimos por hecho que nuestros programas solo se ejecutaban una sola vez, sin la posibilidad no teníamos en cuenta en que el paralelismo puede ser una manera más compleja y a la vez fácil para comprender cómo funciona un sistema operativo o el diseño de redes, aunque sea a una escala demasiado pequeña.

Se nos complico algo porque al compilar los ejercicios en “C” no teníamos idea de porque no nos funcionaba, y en el aspecto de java pensamos que el ejecutable que venía en el libro era código y no simples instrucciones para compilarlo y ejecutarlo en consola.