Laboratorio número 9
Dayana Marín Mayorga.
Carné: B64096

Compilar y correr el programa "omp_hello.c"
-Correr el programa varias veces
-Explicar por qué la salida del programa varía.

```
dayana@dayana-marin:~/Documentos/Tarea-09$
Hello from thread 5 of 7
Hello from thread 4 of 7
Hello from thread 3 of 7
Hello from thread 2 of 7
Hello from thread 1 of 7
Hello from thread 0 of 7
Hello from thread 6 of 7
dayana@dayana-marin:~/Documentos/Tarea-09$
```

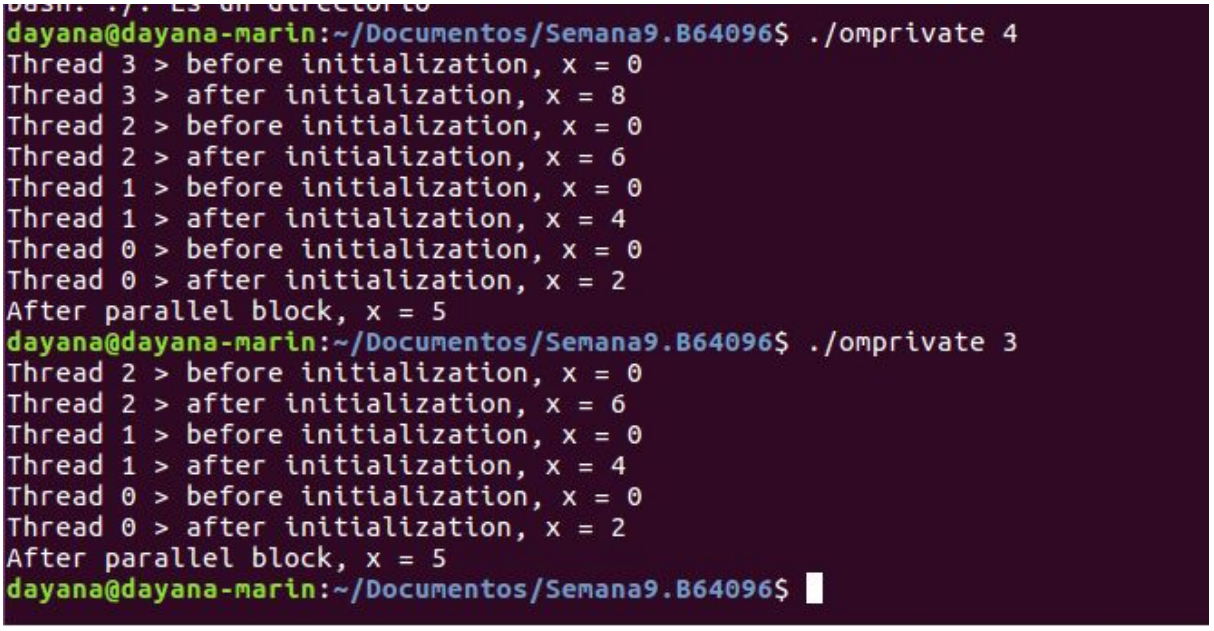
-El comportamiento, y la distribución del trabajo a cada hilo varía.
-Se corre el programa y el primer hilo que termine su funcionamiento será el que se refleja primero en la salida de la consola.
-Cada hilo se mantendrá en espera hasta que el más lento de los hilos termine su funcionamiento para reflejar el resultado.
-También puede ocurrir que un hilo que se imprimió de primero en una corrida, termine siendo el último en la siguiente.

2. Compilar y correr el programa "omp_fibo.c"
-Anotar si el programa despliega el resultado correctamente
- Explique la razón por lo que usted piensa el programa no funciona
- Anote el respaldo teórico de su razonamiento.

```
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./ompfibo 3 7
The first n Fibonacci numbers:
0       1
1       1
2       2
3       3
4       3
5       4
6       2
dayana@dayana-marin:~/Documentos/Semana9.B64096$
```

La razón por lo que el programa no despliega el resultado correcto es por la asignación de hilos, no se asignan números diferentes a todos los hilos, se le están asignando el mismo proceso a todos los hilos, estos corresponden a valores ya calculados que ya un hilo anterior utilizó.

3. Compilar y correr el programa "omp_private.c".
 -Explique el funcionamiento de hacer la variable x privada



-La variable x que se encuentra en el private del pragma solo se modifica dentro del bloque del pragma, entonces su valor no se va a ver modificado, principalmente es para usarla como una variable distinta dentro del pragma.

4. Compilar y correr el programa "omp_trap1.c"
- Corra el programa para comprobar el resultado.
- Escoja valores para los datos y anótalos para las demás pruebas.
- Anote el valor del resultado del programa.

-El a, b corresponden al intervalo de datos del trapezoide, y el n corresponde al número de trapezoides
-La cantidad de trapezoides debe de ser divisible a la cantidad de threads
En la imagen se notan los resultados con los respectivos intervalos threads y trapezoides.

- Elimine "#pragma omp critical"
- Compruebe si el resultado es el mismo
-Comente sobre el problema encontrado

Los resultados cambian y esto se debe a que pragma omp critical identifica la sección de código que tiene que ser ejecutada por un solo hilo al mismo tiempo. Por lo que los valores cambian sin ese pragma critical al no presentarse esta característica.

5. Compilar y correr el programa "omp_trap2a.c"
- Utilice los mismos datos que en el caso anterior
- Anote si el resultado es correcto

```
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./omptrap2a
usage: ./omptrap2a <number of threads>
   number of trapezoids must be evenly divisible by
   number of threads
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./omptrap2a 3
Enter a, b, and n
3 4 9
With n = 9 trapezoids, our estimate
of the integral from 3.000000 to 4.000000 = 1.23353909465021e+01
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./omptrap2a 7
Enter a, b, and n
24 56 21
With n = 21 trapezoids, our estimate
of the integral from 24.000000 to 56.000000 = 5.39430506424792e+04
dayana@dayana-marin:~/Documentos/Semana9.B64096$
```

Los resultados son los mismos.

6. Compilar y correr el programa "omp_trap2b.c"
- Utilice los mismos datos que en el caso anterior
- Anote si el resultado es correcto.

```
dayana@dayana-marin:~/Documentos/Semana9.B64096$
dayana@dayana-marin:~/Documentos/Semana9.B64096$ gcc -g -Wall -fopenmp -I. omp_trap2b.c  -o omptrap2b -lpthread
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ls
bubble.c    omphello        omp_msg          omp_pi.c       omp_sin_sum.c  omptrap2a     omp_trap2b.c
ompfibo     omp_hello.c     omp_odd_even1.c  omp_private.c  omptrap1       omp_trap2a.c  omp_trap3.c
omp_fibo.c  omp_mat_vect.c  omp_odd_even2.c  omprivate      omp_trap1.c    omptrap2b     Read.Me
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./omptrap2b 3
Enter a, b, and n
3 4 9
With n = 9 trapezoids, our estimate
of the integral from 3.000000 to 4.000000 = 1.23353909465021e+01
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./omptrap2b 7
Enter a, b, and n
24 56 21
With n = 21 trapezoids, our estimate
of the integral from 24.000000 to 56.000000 = 5.39430506424792e+04
dayana@dayana-marin:~/Documentos/Semana9.B64096$
```

Los resultados son los mismos.

7. Compilar y correr el programa "omp_trap3.c"
- Utilice los mismos datos que en el caso anterior
- Anote si el resultado es correcto

```
dayana@dayana-marin:~/Documentos/Semana9.B64096$ gcc -g -Wall -fopenmp -I. omp_trap3.c  -o omptrap3 -lpthread
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ls
bubble.c    omphello       omp_msg          omp_pi.c       omp_sin_sum.c  omptrap2a     omp_trap2b.c  Read.Me
ompfibo     omp_hello.c    omp_odd_even1.c  omp_private.c  omptrap1       omp_trap2a.c  omptrap3
omp_fibo.c  omp_mat_vect.c  omp_odd_even2.c  omprivate      omp_trap1.c    omptrap2b     omp_trap3.c
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./ omptrap3
bash: ./: Es un directorio
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./omptrap3 3
Enter a, b, and n
3 4 9
With n = 9 trapezoids, our estimate
of the integral from 3.000000 to 4.000000 = 1.23353909465021e+01
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./omptrap3 7
Enter a, b, and n
24 56 21
With n = 21 trapezoids, our estimate
of the integral from 24.000000 to 56.000000 = 5.39430506424792e+04
dayana@dayana-marin:~/Documentos/Semana9.B64096$
```

Los resultados son los mismos.

8. Compilar y correr el programa "omp_pi.c"
- Correr el programa y comprobar el resultado
-Explicar la manera que se realiza la acumulación de la suma

```
dayana@dayana-marin:~/Documentos/Semana9.B64096$ gcc -g -Wall -fopenmp -I. omp_pi.c  -o omppi -lpthread
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ls
bubble.c    omphello       omp_msg          omppi          omprivate      omp_trap1.c   omptrap2b     omp_trap3.c
ompfibo     omp_hello.c    omp_odd_even1.c  omp_pi.c       omp_sin_sum.c  omptrap2a     omp_trap2b.c  Read.Me
omp_fibo.c  omp_mat_vect.c  omp_odd_even2.c  omp_private.c  omptrap1       omp_trap2a.c  omptrap3
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./omppi 3 6
With n = 6 terms and 3 threads,
   Our estimate of pi = 2.97604617604618
                pi = 3.14159265358979
dayana@dayana-marin:~/Documentos/Semana9.B64096$
```

Se trata sobre un cálculo del número de Pi mediante una serie de potencias.

9. Compilar y correr el programa "bubble.c"
- Correr el programa
-Agregue datos y compruebe el resultado
- Utilice el comando "time" para estimar el tiempo que le toma para realizar el cálculo
- Corra el comando "valgrind ./bubble"
- Interprete la salida y anote los errores que encuentre

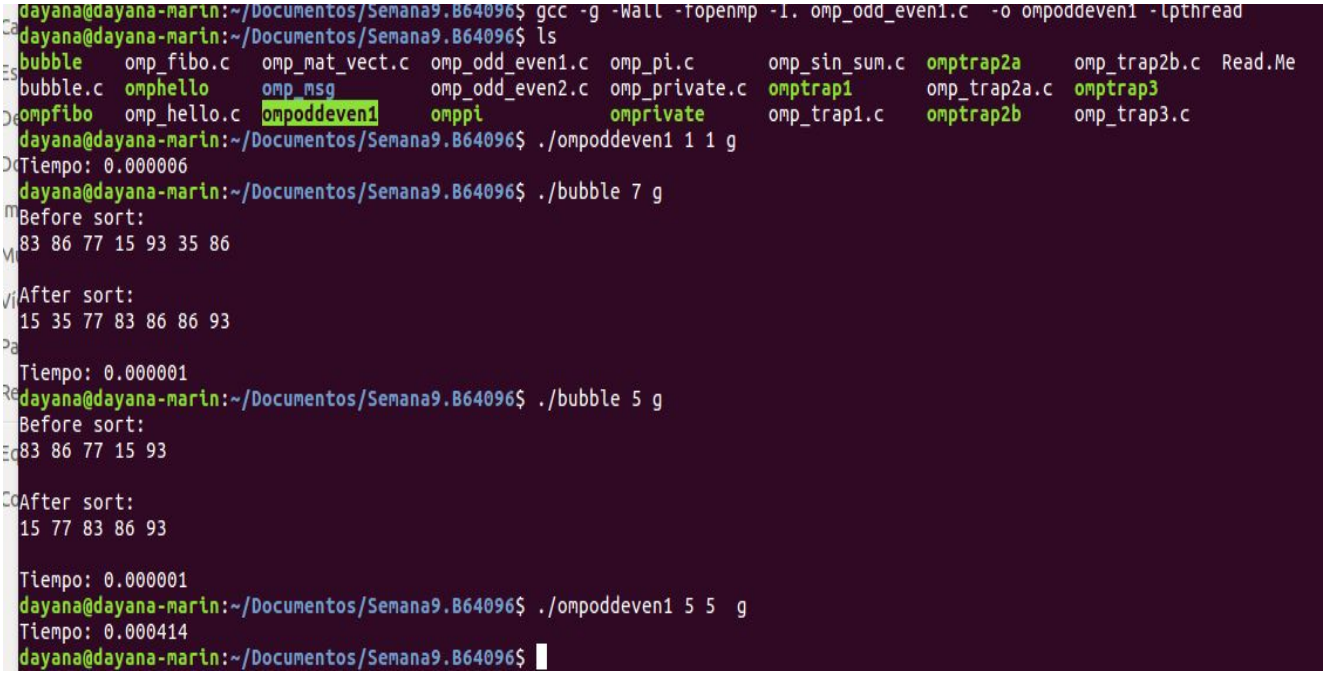
```
dayana@dayana-marin:~/Documentos/Semana9.B64096$ gcc -g -Wall -fopenmp -I. bubble.c  -o bub
ad
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ls
bubble      omphello        omp_odd_even1.c  omp_private.c  omp_trap1.c   omp_trap2b.c
bubble.c    omp_hello.c     omp_odd_even2.c  omprivate      omptrap2a     omptrap3
ompfibo     omp_mat_vect.c  omppi            omp_sin_sum.c  omp_trap2a.c  omp_trap3.c
omp_fibo.c  omp_msg         omp_pi.c         omptrap1       omptrap2b     Read.Me
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./bubble 7 g
Before sort:
83 86 77 15 93 35 86

After sort:
15 35 77 83 86 86 93

Tiempo: 0.000002
dayana@dayana-marin:~/Documentos/Semana9.B64096$ .
```

```
Tiempo: 0.000002
dayana@dayana-marin:~/Documentos/Semana9.B64096$ valgrind ./bubble
==7344== Memcheck, a memory error detector
==7344== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==7344== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==7344== Command: ./bubble
==7344==
usage:   ./bubble <n> <g|i>
   n:   number of elements in list
  'g':  generate list using a random number generator
  'i':  user input list
==7344==
==7344== HEAP SUMMARY:
==7344==     in use at exit: 0 bytes in 0 blocks
==7344==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==7344==
==7344== All heap blocks were freed -- no leaks are possible
==7344==
==7344== For counts of detected and suppressed errors, rerun with: -v
==7344== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
dayana@dayana-marin:~/Documentos/Semana9.B64096$
```

De acuerdo a la última línea, no hay errores a la hora de ejecutar el comando valgrind ./bubble.c.

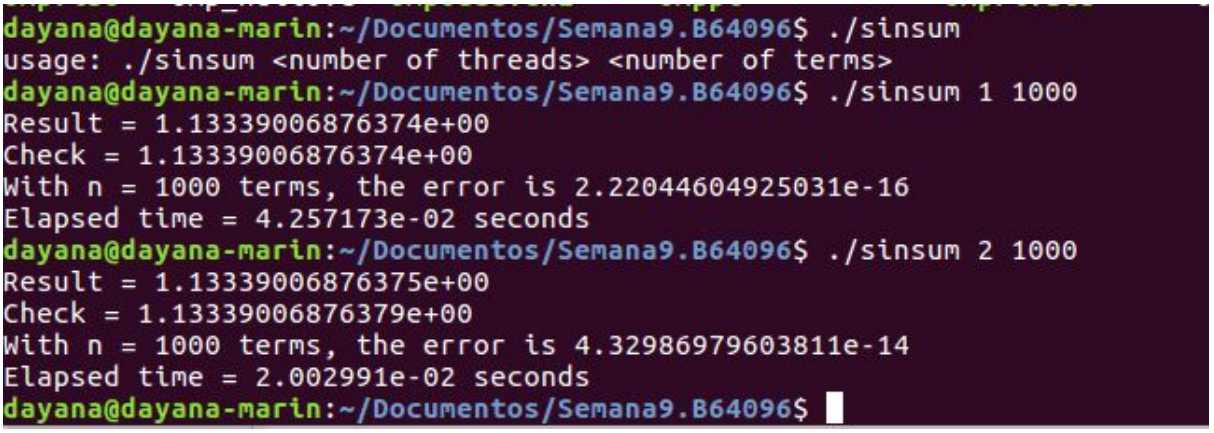
La salida nos indica que valgrind ayuda en la depuración de problemas de memoria y rendimiento de programas.

10. Compilar y correr el programa "omp_odd_even1.c"
-Correr el programa
- Agregue datos y compruebe la salida
- Utilice el comando "time" para estimar el tiempo que le toma para realizar el cálculo, haga una comparación con "bubble.c" y anote los resultados.



Entre 1 a 4 hilos es más rápido que bubbles, pero da resultados más lentos.

11. Compilar y correr el programa "omp_sin_sum.c"
- Correr el programa con 1 hilo y 10 000 términos
- Anote el tiempo
- Correr el programa con 2 hilos y 10 000 términos
- Anote el tiempo
- Calcule el "speedup"
- Note que el programa tiene "schedule(runtime)"
- Escriba en el shell, antes de correr el programa
- OMP_SCHEDULE="dynamic" (pruebe con "auto" y "static" también)
- Correr el programa de nuevo y anote las diferencias en los tres casos indicados.



Serial un hilo: 4.257173e-02
Pthreads dos hilos:2.002991e-02
SpeedUp: 2.125409014

12. Cambiar al directorio "omp_msg"
- Compilar y correr "omp_msgps.c"
- Analice la salida y describa qué está sucediendo
- Busque e indique la funcionalidad de "barrier" y "atomic" en este programa

```
p_msgps 3 4
Thread 2 > received -2 from 2
Thread 2 > received 0 from 0
Thread 0 > received 0 from 2
Thread 0 > received -3 from 2
Thread 0 > received -1 from 0
Thread 0 > received -2 from 0
Thread 0 > received -3 from 0
Thread 1 > received -1 from 2
Thread 1 > received 0 from 1
Thread 1 > received -2 from 1
Thread 0 > received -1 from 1
Thread 0 > received -3 from 1
```

- La asignación de los hilos se hace a través de un srandom, lo que hace que haya hilos que tengan más de un proceso, y otros que no tengan procesos.
-La funcionalidad del barrier no permite que ningún thread mande mensajes hasta que todas las colas estén construidas.
-La funcionalidad de atomic lo que hace es permitir el acceso de una posición de memoria específica de manera atómica, lo que asegura que no se den las condiciones de carrera porque permite un control directo de los hilos de concurrencia, mediante lectura o escritura..

13. Compilar y correr el programa "omp_mat_vect.c"
- Correr el programa y anotar los datos de corrida
- Cambiar alguno de los parámetros y anote los resultados

```
omp_hello.c  omppi          omp_trap1.c    omp_trap3.c
dayana@dayana-marin:~/Documentos/Semana9.B64096$ gcc -g -Wall -fopenmp omp_mat_vect.c  -o ompmatvect -lpthread
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ls
bubble      ompmatvect      omppi         omp_trap1.c   omp_trap3.c
bubble.c    omp_mat_vect.c  omp_pi.c      omptrap2a     Read.Me
ompfibo     omp_msg         omp_private.c  omp_trap2a.c  sinsum
omp_fibo.c  ompoddeven1     omprivate     omptrap2b
omphello    omp_odd_even1.c omp_sin_sum.c  omp_trap2b.c
omp_hello.c  omp_odd_even2.c  omptrap1      omptrap3
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./ompmatvect 3 4 5
Elapsed time = 1.456030e-04 seconds
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./ompmatvect 4 5 6
Elapsed time = 1.566940e-04 seconds
dayana@dayana-marin:~/Documentos/Semana9.B64096$ ./ompmatvect 5 6 7
Elapsed time = 1.948140e-04 seconds
dayana@dayana-marin:~/Documentos/Semana9.B64096$
```

Cambia el tiempo y mejora conforme a la cantidad de threads.