Universidad del Valle de Guatemala
Sistemas Operativos
Tomás Gálvez

Ivette Cardona
Carné 16020
Fecha: 25/03/19

Laboratorio 5

- **¿Qué es una race condition y por qué hay que evitarlas?**
  Sucede cuando un dispositivo o sistema intenta realizar dos o más operaciones al mismo tiempo. Se deben evitar ya que debido a la naturaleza del dispositivo a sistema, las operaciones deben realizarse en una secuencia apropiada para que se ejecuten correctamente, de lo contrario pueden afectar de manera negativa a este.

- **¿Cuál es la relación entre pthreads y clone()? ¿Hay diferencia al crear threads con uno o con otro?¿Qué es más recomendable?**
  clone() crea un nuevo proceso, parecido al funcionamiento de fork(). En cambio, con pthreads_create() se crea un nuevo thread dentro del proceso donde fue llamado. Pocas personas utilizan clone() directamente ya que es más portable utilizar la librería pthreads.

  La llamada de sistema clone() crea un nuevo contexto de ejecución, o COE. Este puede ser un nuevo proceso, un nuevo thread, dependiendo del tipo de uso que se le desee dar.

- **¿Dónde,en su programa,hay paralelización de tareas, y dónde de datos?**
  - Tareas: se caracteriza por el hecho de tener diferentes operaciones que se realizan sobre los mismos o diferentes datos.
  - Datos: se diferencia porque las mismas operaciones se utilizan sobre diferentes subsets de la misma data.

  Se paraleliza en el programa por medio de OpenMP y multithreading por medio de pthreads. OpenMP se inicializa en la sección del pragma omp

  ```
  >>-#--pragma--omp parallel
  ```

  Y los threads al momento de inicializar la cantidad de threads e indicarles sobre qué datos se deseaban paralelizar.

- **Al agregar los #pragmas a los ciclos for, ¿cuántos LWP's hay abiertos antes de terminar el main()y cuántos durante la revisión de columnas?¿Cuántos user threads deben haber abiertos en cada caso, entonces?**
  Hint:recuerde el modelo de multithreading que usa Linux.
  Se tienen solo 1 LWP´s y 11 threads. Es recomendable tener un thread para cada acción a realizar (Ej: verificar las columnas).

- **Al limitar el número de threads en main()a uno, ¿cuántos LWP's hay abiertos durante la revisión de columnas? Compare esto con el número de LWP's abiertos antes de limitar el número de threads en main(). ¿Cuántos threads crea OpenMP (en general) por defecto?**

  Hay 8 LWP´s durante la revisión de columnas y antes de limitar la cantidad de threads había solo 1.
  OpenMP elige un número óptimos de threads por default, este número es usualmente el número de cores existentes. OpenMP automáticamente distribuye las iteraciones entre los threads y si la división que se realiza no es par, entonces un thread termina haciendo más o menos trabajo que los demás.

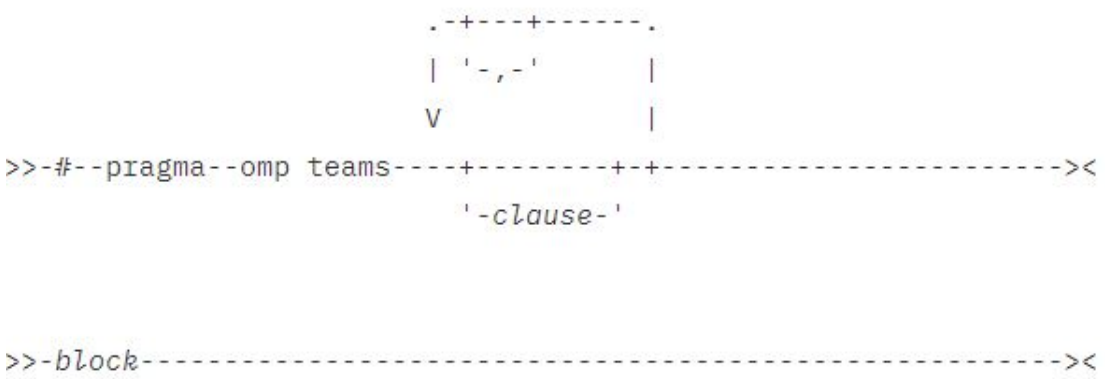
- **Observe cuáles LWP's están abiertos durante la revisión de columnas según ps. ¿Qué significa la primera columna de resultados de este comando? ¿Cuál es el LWP que está inactivo y por qué está inactivo?**

  LWP (Light Weight Process), el kernel puede distinguir entre los procesos completos y aquellos que se ejecutan paralelamente por medio de threads.
  La primera columna hace referencia al proceso creado en donde los threads estarán realizando las operaciones.

```
  LWP
1424
1481
1482
1483
1484
1485
```

- **Compare los resultados de ps en la pregunta anterior con los que son desplegados por el método de revisión de columnas per se.¿Qué es un thread team en OpenMP y cuál es el master thread en este caso?¿Por qué parece haber un thread"corriendo", pero que no está haciendo nada? ¿Qué significa el término busy-wait?¿Cómo maneja OpenMP su thread pool?**

La directiva omp teams crea una colección de *thread teams.* El master thread de cada team es el encargado de ejecutar la región otorgada al team.



En este caso el thread master es el que se inicializó para empezar a revisar las columnas (9059) y llegó hasta que el proceso terminó de ejecutarse. Este es el thread que pareciera que no hace nada pero sigue corriendo, ya que es una serie de instrucciones consecutivas, quien crea un fork() para dar lugar a que trabajen los slave threads.

El término busy-wait implica que un thread debe esperar a que se cumpla una condición o si el ingreso a una sección crítica está habilitado para poder realizar la tarea que se le encomendó.

```
int nThreads;
#pragma omp parallel
nThreads = omp_get_num_threads();
```

Esto se realiza con la intención de preparar el thread pool de OpenMP con un complemento lleno de threads. El thread pool sirve para tener listos los threads que se utilizarán a lo largo del proceso, de manera que OpenMP establece si la cantidad especificada por nosotros es adecuada para el proceso que se desee verificar.

- **Luego de agregar por primera vez la cláusula schedule(dynamic)y ejecutar su programa repetidas veces, ¿cuál es el máximo número de threads trabajando según el método de revisión de columnas? Al comparar este número con la cantidad de LWP's que se creaban antes de agregar schedule(), ¿qué deduce sobre la distribución de trabajo que OpenMP hace por defecto?**

- **Luego de agregar las llamadas omp_set_num_threads() a cada método donde se usa OpenMP,y luego de ejecutar su programa con y sin la cláusula schedule()en cada for, ¿hay más o menos concurrencia en su programa? ¿Es esto sinónimo de un mejor desempeño? Explique.**
  Implica mayor concurrencia pero no mejor desempeño.

- **¿Cuál es el efecto de agregar omp_set_nested(true)? Explique.**
  Es para poner Nested Parallelism, esto permite crear una región paralela a la ya existente (una especie de paralelismo anidado). Además, también permite ajustar el tamaño de los threads teams.