1.¿Qué es una race conditiony por qué hay que evitarlas?

Una race condition se refiere a lo que ocurre cuando dos o más threads tienen acceso simultaneo a la misma data y éstos tratan de modificarla al mismo tiempo. Es importante evitar estos eventos, porque como el sistema operativo puede cambiar entre threads en cualquier momento, se vuelve difícil saber cuál thread modificará la data primero, haciendo que nuestro programa no se comporte como nosotros queremos.

2.¿Cuál es la relación, en Linux,entre pthreadsy clone()? ¿Hay diferencia al crear threadscon uno o con otro?¿Qué es más recomendable?

Ambas funciones pueden ser utilizadas para crear threads, pero clone tiene la capacidad de crear procesos también. La diferencia entre crear threads con estas funciones es que clone también crea procesos, lo cuál ocupa más memoria. Es más recomendable utilizar pthreads para crear threads y una de las razones principales es que clone no es portable.

3.¿Dónde,en su programa,hay paralelización de tareas, y dónde de datos?

En nuestro programa existe paralelización de tareas en los ciclos for que asignamos a varios procesadores, porque la tarea es la misma, mientras que los datos cambian.

4.Al agregar los #pragmas a los ciclos for, ¿cuántos LWP’shay 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 usan Linux y Windows.

Linux utiliza el modelo de multithreading uno a uno, por lo que esperaríamos encontrar la misma cantidad de user threads como de LWP´s. Al terminar el main hay 6 LWP´s, mientras que durante la revisión de columnas hay 7.

5.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 threadsen main(). ¿Cuántos threads(en general) creaOpenMP por defecto?

Al limitar el número de threads el programa abre un solo LWP. Pero antes de limitarlo también se abre un solo LWP, por lo tanto, en general OpenMP abre un thread.

6.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?Hint: consulte las páginas del manual sobre ps.

La primera columna es una variable binaria que nos indica si el thread está activo o no. El LWP que está inactivo es el thread del master y es porque su ID es 0.

7.Comparelos resultados de psen la pregunta anterior con los que son desplegados por la funciónde revisión de columnas per se.¿Qué es un thread teamen OpenMP y cuál es el master threaden 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?

8.Luego de agregar por primera vez la cláusula schedule(dynamic)y ejecutar su programa repetidas veces, ¿cuál es el máximo número de threadstrabajando según la funciónde 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?

9.Luego de agregar las llamadas omp\_set\_num\_threads()a cada función donde se usa OpenMP y probar su programa, antes de agregar omp\_set\_nested(true), ¿hay más o menos concurrencia en su programa? ¿Es esto sinónimo de un mejor desempeño? Explique.

10.¿Cuál es el efecto de agregar omp\_set\_nested(true)? Explique.

Agregar esa línea de código permite que haya nested paralelism, lo cuál hace que se creen más threads que antes.