Cristopher Jose Rodolfo Barrios Sollis 18207

1. ¿Qué es una race condition y por qué hay que evitarlas?
   Una race condition se refiere a que cuando muchos procesos acceden a muchos datos al mismo tiempo, el resultado que da al final de toda esta ejecución da según el orden de esos procesos.
   ¿Por que hay que evitarlas? Como no se ejecutan al mismo orden, sino que de un modo mas o menos aleatorio este puede dar resultado distintos.

2. ¿Cuál es la relación, en Linux,entre pthreads y clone()? ¿Hay diferencia al crear threads con uno o con otro?¿Qué es más recomendable?
   Pues es un poco extraño con linux con respecto a los threads, ya que para crear threads se tiene que utilizar la llamada al sistema clone() esta herramienta lo que hace es enviarlo a un conjunto de indicadores que determinan cuanto se debe compartir entre las tareas primarias y secundarias.

   Los pthreads llevan una serie de pasos, para crear uno hay que declarar una variable tipo pthread_t, luego llamar la función pthread_create() que toma el puntero al pthread_id atributos, luego hay que ponerle el nombre del pthread y el otro parametro pasa argumentos a la función.

3. ¿Dónde,en su programa,hay paralelización de tareas, y dónde de datos?
   Tenemos paralelización cuando se crean pthreads, cada thread ejecuta una tarea diferente, dentro de cada for tenemos paralelismo gracias a OpenMP

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 abierto en cada caso, entonces?Hint:recuerde el modelo de multithreading que usan Linux y Windows.
   Linux y Window usan el modelo multithreading 1 a 1, esto habla de que mapea cada thread a un kernel thread. 5 threads durante la evaluación y al final uno.

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 threads en main(). ¿Cuántos threads(en general) creaOpenMP por defecto?
   Ya solo se tiene a OpenMP, LWP, entonces OPENMP los que hace es encontrar la cantidad de procesadores y según cuantos haya pone la misma cantidad de threads, pero si se puede manipular cuantos threads se requieren.

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.
   Para conocer el estado del proceso se tiene que utilizar el comando ps, para ver la columna S. Los que tienen R están activos Y las S están detenidos.

7. Compare los resultados de psen la pregunta anterior con los que son desplegados por la función de revisión de columnas per se.¿Qué es un thread teamen 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?
Cuando se tiene OpenMP esta crea equipos de threads que al final se ejecutan a la vez, el master de cada equipo ejecuta la region de equipos. El thread que aparece corriendo y no hace nada es aquel que está encima de otros.

8. 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 la función 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?
   El OpenMP por defecto crea 4 threads que es el máximo que este tiene, osea que esto busca hacer la manera más óptima posible su trabajo.

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.
   Al principio tiene una mayor concurrencia, luego se implementa omp_set_nested(true), des pues ya se tienen los threads agrupados y se ordenan. sin embargo los threads implican que consuman mas tiempo y memoria por lo que no puede ser tan eficiente.

10. ¿Cuál es el efecto de agregar omp_set_nested(true)? Explique.
    Como se decía anteriormente, que este tiene un mejor ordenamiento de threads pero esto puede llegar a consumir mas tiempo y memoria porque este no posee cierto orden, porque ejecuta los threads en el momento mas oportuna, si está en grupo si lleva cierto orden y esta obligado a ejecutar el thread como se indica.