1. **¿Qué es una race condition y por qué hay que evitarlas?**

Se le llama así cuando sucede que más de un thread esta tratando de acceder a recursos en común, pero el orden de los accesos puede varía dependiendo del progreso de cada thread. Sin estas no son intencionadas, pues se les puede considerar como un error de programación. Estas hay que evitarlas, porque dependiendo de lo que estemos haciendo, un fallo de sincronización puede ser fatal y afectar a alguien.

1. **¿Cuál es la relación, en Linux, entre pthreads y clone()? ¿Hay diferencia al crear threads con uno o con otro? ¿Qué es más recomendable?**

Al final ambos son lo mismo, pues la implementaciond de pthreads en lines usa a clone, por lo que no importa cual se use. La diferencia recae en que los pthreads son parte del estándar de POSIX.

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

Paralizacion de tareas son los mismos threads con los ciclos y el resto del código es de datos sobre cada una de las iteraciones con el mismo código, pero en diferentes threads, sobre diferentes threads.

1. **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 usan Linux y Windows.**

Ambos modelos de multithreading son de uno a uno, por lo que al existen 4 LWPs y durante las revisiones de columnas son 8. Al final son 12 user threads

1. **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) crea OpenMP por defecto?**

En general se crea la misma cantidad de threads como de procesadores, pero LWP abiertos hay 5 durante la revisión de columnas.

1. **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 de las columnas muestra banderas numéricas sobre la información de si un proceso está haciendo uso de privilegios del tipo superuser. También si existe alguna relación padre-hijo de los procesos creados. El LWP que esta inactivo es del main y esta así porque está a la espera de que se complete la revisión de columnas.

1. **Compare los resultados de ps en la pregunta anterior con los que son desplegados por la función de revisión de columnas per se. ¿Qué es un thread team en 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?**

El thread team es lo mismo que el pool de threads, pero openMP lo llama diferente. El thread mater será el thread encargado de las instrucciones de paralelización. Los threads o threads puede mostrarse corriendo, pero el mismo OpenMP provee el pool de threads, pero no garantiza el uso de todos. El busy-waiting se realiza cuando thread o proceso probando una condición repetidamente para mantenerse activo, por lo que se asigna trabajo. OpenMP abre el thread pool y asigna el estado a los threads y una política de espera determina el tiempo que permanecerán trabajando hasta sus suspensiones.

1. **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 máximo es de 4 y se puede ver como OpenMP implementa un balanceo de carga por defecto a cada thread.

1. **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.**

No es exactamente una mejora del todas, ni tampoco un sinónimo en el sentido literal, pues por el overhead incurrido en la organización de los threads puede opacar dicho beneficio al distribuir el trabajo. Claro que existe concurrencia, porque se aumenta el numero de threads en ejecución.

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

Esto permite la paralización añadida, la cual no se encuentra actividad por el sistema por defecto. Ya con esto se ejecutarán los threas que el ciclo contuviese, aunque se viera precedido por un pragma al paralelizar.