UNIVERSIDAD DEL VALLE DE GUATEMALA

Sistemas Operativos

Laboratorio No.

Isabella Miralles #22293

Guatemala, 2025

1. **¿Qué es una race condition y por qué hay que evitarlas?**
    - Es cuando dos o más threads acceden y modifican datos compartidos de forma concurrente sin la sincronización adecuada, lo que puede producir resultados impredecibles. Debe evitarse para garantizar la integridad y consistencia de los datos
2. **¿Cuál es la relación, en Linux, entre pthreads y clone()? ¿Hay diferencia al crear threads conuno o con otro? ¿Qué es más recomendable?**
    - Los pthreads se implementan internamente usando la llamada al sistema clone(), la cual permita crear procesos ligeros. Aunque se puede usar clone() directamente, el uso de pthreads es más estándar, portable y recomendable por su abstracción y soporte en bibliotecas.
3. **¿Dónde, en su programa, hay paralelización de tareas, y dónde de datos?**
    - La paralelización de tareas es la creación de threads y la ejecución concurrente de forks para ejecutar el comando ps.
    - La paralelización de datos son los ciclos for paralelos que recorren filas, columnas y subarreglos para la validación de sudoku.
4. **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.**
    - El numero de hilos a nivel de usuario es igual al número de LWP que se observa en ps -lLf para un proceso dado.
    - Antes de terminar main() se observa 1 LWP para el hilo principal, dependiendo de la implementación de OpenMP y si mantiene untread pool en reposo, podrían seguir viéndose varios hilos inactivos.
    - 1 LWP para el proceso principal, 1 LWP para el pthread que invoca la revisión de columnas y N LWP adicionales creados por OpenMP para paralelizar el for. En total son 6 LWP y 6 user threads a nivel de programación.
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) crea OpenMP por defecto?**
    - Cuando se limita el número de threads a 1
      El hilo principal sigue existiendo
      Al crear el pthread para revisar columnas, se añade otro LWP.
      Dentro de la región paralela es posible que OpenMP use 1 hilo o si en esa función no se fuerza un número de hilos mayor, no haya paralelismo adicional.
      Se ven 2 LWP en total durante la revisión de columnas.

- Antes de limitar el número de threads a 1
  OpenMP crea un número de hilos igual al número de núcleos lógicos de la CPU.
  Del hilo principal el pthread también lanza la región paralela de columnas, creando 4 hilos OpenMP.
  Se puede ver 1 main, 1 pthread, 4 OpenMP, que son 6 LWPs en ese instante.
- OpenMP crea tantos hilos como núcleos lógicos detectados, a menos que se establezca la variable de entorno OMP_NUM_THREADS o se use la llamada omp_set_num_threads() para forzar otro número.

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.**
   - El hilo principal puede estar esperando a que el pthread termine, o algún hilo OpenMP puede estar en espera mientras otro hilo hace el trabajo.
7. **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 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?**
   - El thread team en OpenMP es cuando entra a una región paralela, se crea un equipo de hilos formado por el master thread y los worker threads.
   - El master thread es el hilo que ejecuta el bloque de código fuera de la región paralela y que al entrar a #pragma omp parallel, se convierte en el master dentro del equipo.
   - En ocasiones un hilo se ve en estado running pero en realidad está en un busy-wait, esperando que lleguen tareas o que otro hilo termine. OpernMP suele usar busy-wait en barreras y en threads pools para reaccionar rápido a nuevo trabajo, en vex de dormir y despertar con interrupciones.
   - Thread Pool en OpenMP
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?**
   -
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.**
    -