Universidad del Valle de Guatemala
Sistemas Operativos

Eduardo Ramírez Herrera
19946

Laboratorio #3

1. **¿Qué es una race condition y por qué hay que evitarlas?**
   Una race condición es cuando mas de un hilo accede al mismo recurso a la vez, esto puede pasar cuando dos hilos tienen la intención de acceder al mismo espacio de memoria y uno de los dos hilos logra sobre escribir la información, debemos tratar de hacer que no existan estas race condition realizando un buen manejo de hilos para que los resultados sean esperados y no existan estos casos que nos pueden dar errores en los programas.

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?**
   Los Pthreads los podemos utilizar entre distintos sistemas ya que así esta definido dentro del estándar de POSIX, no existen diferencias al crear varios hilos con hilos de procesos o incluso con la función de clone porque la implementación de los pthreads para los sistemas de Linux utiliza la función de clone().

3. **¿Dónde, en su programa, hay paralelización de tareas, y dónde de datos?**
   En el programa existe paralelización a nivel de tareas por medio de ciclos for de manera de que estos realicen operaciones de asignación, validación y algunas aritméticas. También existe paralelización al momento de realizar validaciones para que dentro de arrays exista por lo menos un objeto, en este caso seria un numero del 1 al 9, esta es una instrucción que se realiza constantemente con una gran cantidad de datos simultáneos.

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.**

   Al terminar el main() hay 4 LWP's, sin embargo, durante la revisión de columnas tenemos 8. Esto quiere decir que hay 4 y 8 user threads abiertos respectivamente.

Universidad del Valle de Guatemala
Sistemas Operativos

Eduardo Ramírez Herrera
19946

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?**

   Si limitamos el numero de threads dentro del main() automáticamente pasamos a tener únicamente un solo thread, antes de esto se contaba con 4 threads. Normalmente se crean threads que van acorde a la cantidad de los procesadores que están disponibles.

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.**

   En la columna S se indica el estado en el que se encuentra el proceso por lo que podemos decir que los 4 lwp se encuentran activos R y uno esta inactivo S. Ya que existe un LWP inactivo quiere decir que este es el proceso que dio vida y origen a otros 4 procesos, por lo que este es el proceso padre

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?**

   Podemos observar que los procesos activos son los mismos. Un conjunto de threads se denomina un team of threads y estos están listos para ser ejecutados cuando sea necesario dependiendo el numero de procesadores. El master thread es el que se ejecuta sin realizar nada sobre las operaciones y esto ocurre porque este se encarga de controlar que otros threads realice todas sus asignaciones. OpenMP abre las pool threads donde cada uno de los threads esta corriendo. Al momento de que los threads del kernel hayan sido creados se termina el trabajo y regresa a un dock por las tareas nuevas.

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?**

   Se hicieron 3 threads en la revisión de cada una de las columnas y comparando con cada uno de los 4 de antes podemos observar que OpenMP distribuye las tareas para poder usar los procesadores que están disponibles. Esto lo hace de manera dinámica y se reduce el numero de threads porque se valida si necesario usarlos para la tarea que se esta realizando.

Universidad del Valle de Guatemala
Sistemas Operativos

Eduardo Ramírez Herrera
19946

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.**

   Podemos incrementar el numero de threads en ejecución y esto lo que ocasiona es que exista mas concurrencia, pero tampoco significa que exista una mejora en el desempeño porque cada instrucción debe esperar el momento en el que llegue su turno para poder utilizar los datos. Mediante el overhead el organizar threads no puede ser lo mars beneficioso.

10. **¿Cuál es el efecto de agregar omp_set_nested(true)? Explique.**

    Esto hace que haya una paralelización anidada por lo que los ciclos aninados creearan sus propios threads y esto hace que exista mas LWPS.