Sergio Marchena
16387

LAB 5

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

Una race condition es cuando, por ejemplo, dos threads diferentes desean escribir en un mismo espacio de memoria compartida. El problema sería que el primer thread cambiara el valor del registro, entonces al momento de qué el segundo thread quiera acceder y utilizarlo no se tendrán los resultados esperados porque el valor ya no era el mismo que al principio. Para evitarlas existen varios mecanismos, como bloquear la parte de escritura hasta que un thread la use y después desbloquearla.

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?

Clone crea un nuevo proceso, y deja que este comparta partes de su contexto de ejecución con el padre, como espacio de memoria. Esta característica la comparten también los threads; clone es normalmente utilizado para implementar threads. Clone() es utilizado tanto por fork, demás y pthreads al final de cuentas; entonces, podríamos decir que clone es la función que unifica threads y procesos. Las diferencias percibidas entre threads y procesos se logran pasando diferentes banderas al syscall clone (). Y cuando es mejor utilizar una que otro es determinando exactamente qué es compartido por los espacios de memoria y decidir si es mejor utilizar procesos o threads.

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

La paralelización de tareas existe cuando asignamos el método de checkColumns, checkRows o checkSubsec para que sean ejecutados. El paralelismo de datos se da cuando paralelizamos los ciclos for, esto es notorio ya que el mismo laboratorio nos advierte de las race conditions.

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 multi threading que usan Linux y Windows.

Si no se especifica el número de threads se crearán tantos como CPU's lógicos existan en el procesador.

5. Compárelos 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?

Cuando iniciamos la creación de threads para paralelismo especificamos los threads que van a encargarse de una región en específico. Este será nuestro equipo de threads, donde el master thread es el thread principal desde donde fueron creados los demás. Una busy-wait es cuando esperamos a que termine un thread, haciendo un ciclo que no haga nada, más que acaparar tiempo de procesador en lo que otro thread termina para asegurar sincronización. El thread pool es cuantos threads tengo creados ya para realizar la paralelización en distintas regiones. La manera en que OpenMP trbaja es

que si yo hay 8 threads para una primera región, y se indica que se utlizaran 4 en la segunda región, 4 threads se quedarán dormidos.

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

Hacer paralelismo dentro de una región ya paralelizada. Así, hace posible que un thread que corra sobre una región pueda instanciar más threads cuando encuentra la otra región paralelizada.