PREGUNTAS:

1. Ejecutar los siguientes casos y justificar su comportamiento:

a. ./practica1 (multicore)

El programa ejecuta con normalidad con costes cercanos a 0,5 y respetando siempre el periodo de 0.9 segundos, sin dar ningún fallo temporal en ninguno de los threads ni ninguna de las iteraciones. Al disponer de 12 núcleos los 4 hilos de ejecución concurrente pueden ejecutar sin ningún problema ya que el procesador no está prácticamente ocupado.

b. ./practica1 (monocore)

En este observamos que al solo disponer de un núcleo y los procesos tener la necesidad de ir entrando al core es imposible que todos los procesos ejecuten durante 0,5 segundos en un periodo de 0,9 ya que con solo dos procesos necesitamos mínimo un periodo de un segundo para poder ejecutar ambos procesos.

En este caso obtengo fallo temporal en todos los threads y en todas las iteraciones lo que me parece normal ya que el núcleo no espera a terminar cada acción de cada thread si no que los gestiona para intentar que todos terminen.

Con htop observo que sin la opción stress el procesador ya se utiliza al 100%.

c. ./practica1 (monocore) + stress

Obtengo resultados muy parecidos al apartado anterior ya que al usar solo un core en condiciones normales el uso de la cpu ya es de un 100% por lo que no se nota gran cambio estresando la cpu. Sigo obteniendo fallo temporal en todas las iteraciones y threads.

d. ./practica1 (multicore) + stress

Utilizando el comando stress para todos los núcleos del ordenador obtenemos una ejecución mixta entre los casos anteriores. En diferentes ejecuciones o siempre obtenemos el mismo resultado obteniendo algunos fallos temporales aunque no en todos los threads ni en las mismas iteraciones. A pesar del elevado uso de las cpus al disponer de 12 cpus en la primera prueba consigo varias ejecuciones sin fallo temporal.

Al probar este mismo comando en el ordenador indicado con 6 núcleos obtengo el resultado esperado con fallos temporales en casi todas las iteraciones debido a la cercanía entre número de núcleos y procesos ejecutando y el poquito tiempo de ejecución que el kernel puede meter estos procesos debido al comando stress. Al disponer de menos núcleos se consigue un peor resultado ya que en un mismo periodo de tiempo no pueden ejecutar la misma cantidad de veces.

2. ¿En qué casos de ejecución (nombrados anteriormente) el sistema es capaz de cumplir las restricciones temporales (tanto tiempo de cómputo como periodicidad)?

En el ordenador de pruebas el único caso de ejecución que puede cumplir con ambos requisitos sería la ejecución normal con multicore. Aun así en mis prueba en el ordenador designado se cumple siempre la periodicidad pero no siempre el tiempo de cómputo a pesar de contar con más núcleos que que threads de ejecución al ser 4 thread y tan solo 6 núcleos es posible que por otros proceso que se estén ejecutando u otras personas que puedan estar usando el ordenador de forma remota y empeoren la ejecución.

En unas condiciones optimas pienso que deberia poder ejecutar con valores de computo muy cercanos a 0,5 teniendo en cuenta que los maximos valores de computo obtenidos son de 0,6.

3. ¿Qué número mínimo de cpus se necesitan para que tu programa ejecute correctamente sin fallos de restricciones temporales? Usa el comando taskset para comprobarlo.

En condiciones ideales, el programa ejecuta correctamente con 3 nucleos minimo disponibles dado que el coste de computo es menor que los periodos por lo que con los huecos restantes en los que un thread no ejecuta puede ejecutarse otros supliendo la falta de un core a base de esos momentos en los que ya ha ejecutado.

A pesar de que el periodo se cumple los costes sí que aumentan levemente al disponer de menos recursos y no poder ejecutar libremente.

4. ¿Qué solución se podría proponer para cumplir plazos estrictos temporales de periodicidad en la ejecución de los threads SIN cambiar la configuración actual que tienen los ordenadores del laboratorio?

La solución posible que se me ocurre sería la utilización de prioridades, en este caso con un comando el comando de la shell nice que nos permite darle una prioridad más alta al prioridad al proceso que vamos a ejecutar permitiendo así que nuestros threads ejecuten más tiempo en el intervalo que dura su periodo ya que el planificador del sistema les va a dar cierta prioridad respecto al resto de procesos.