Memoria Práctica 1.

1.

a) ./practica 1 (multicore)

Al ejecutar de esta manera el código, se puede observar que la gran mayoría de las ejecuciones, las iteraciones cumplen el *deadline* establecido. Esto debido a que al ejecutarse todas las CPUs del sistema sin estar muy usadas por otras tareas, permiten distribuir las tareas entre los distintos procesadores. Esto lo lleva a cabo el scheduler en el kernel.

b) ./practica1 (monocore)

Al ejecutarlo con el sistema en monocore, siempre da fallo temporal. Tarda aproximádamente más de 2 segundos en ejecutar cada conjunto de iteraciones. Esto se debe a que el sistema ejecuta todos los threads con un solo procesador, es decir, como un sistema monotarea con un único bucle de ejecución, siendo imposible distribuirse las tareas y poder realizar paralelismo o que se produzca concurrencia.

c) ./practica1 (monocore) + stress

Como se ha visto en el apartado anterior, al ejecutarse como un sistema con una sola CPU, se producirá fallo temporal, ya que si además de ejecutar varios hilos de ejecución en un solo procesador, le añadimos estrés computacional, este tendrá que planificar las tareas, no respetando el *deadline.*

```
taskset -c 0 ./practica1
```

d) ./practica1 (multicore) + stress

Para ejecutar esta acción, se ejecutará este comando en una terminal distinta a la que ejecuta el programa:

```
stress -c 2
```

En este caso, se producirán varios fallos temporales, pero no muchos, debido a que se han aumentado el número de CPUs que ejecutarán este programa, pero a dos de ellos se les ha añadido estrés computacional, aumentando así la latencia en varias de sus iteraciones. Si se añadiera más estrés, cada vez habrá menos iteraciones que cumplan con el *deadline*.

2.

En el único caso en el que en mi sistema se cumple con el deadline es en el de multicore sin estrés. En todos los demás, al menos 1 vez se produce siempre un fallo temporal.

3.

Para averiguar el número de CPUs necesarias para para que no haya fallos temporales, se ha ido ejecutando este comando de manera sucesiva:

```
taskset -c 0 ./practica1 → monocore, siempre hay fallo temporal.
taskset -c 0,1 ./practica1 → Empieza a ver menos fallos.
taskset -c 0,1,2 ./practica1 → Cada vez hay menos fallos temporales.
taskset -c 0,1,2,3 ./practica1 → Con 4 procesadores ya no hay fallos temporales.
```

Luego, ya no hay fallos temporales a partir de 4 procesadores.

4.

La solución más obvia, sería aumentar el *deadline* de las iteraciones, pero ¿hasta cuándo? No es una solución valida.

Tampoco se podría utilizar un planificador Rate Monotonic, ya que haciéndo los cálculos, $\boldsymbol{U > 1}$, por tanto no se garantiza que se cumplan los deadlines.

Una de las soluciones que se podría proponer sería que se utilizaran locks en los threads, para así se ejecuten de uno en uno de forma lineal, evitando que se llenen las CPU de procesamiento y cumplir así con los deadlines establecidos. Actuaría como si no existiera paralelismo e iría soltando el lock una vez termine cada thread.

Jaime Avilleira García