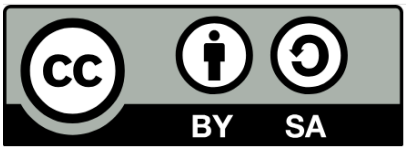
Universidad
Rey Juan Carlos

Grado de Ingeniería en Robótica Software

Sistema empotrados y de tiempo real

Práctica 2

Cácuulo de latencias en sistemas RT y no-RT



En esta segunda práctica pondremos en práctica los conceptos adquiridos en clase de teoría sobre cálculo de latencias y sistemas de tiempo real.

1. Análisis con cyclictest.

Utiliza la utilidad cyclictest para medir la latencia en los laboratorios de la universidad, tanto en ordenadores con kernel normal como en ordenadores con el kernel de RT. Utiliza el siguiente comando, pero **asegúrate** de mirar la página de manual para entender los parámetros utilizados.

```
$ sudo cyclictest -p99 -N --smp -D 60
```

La medición de la tenencia siempre se debe realizar en 3 escenarios distintos:

- S1: idle
- S2: Estresando el planificador (hackbench)
  - `sudo hackbench -l 1000000 -s 1000 -T `nproc``
- S3: Estresando las operaciones I/O (bonnie++)
  - `bonnie++ -d /tmp/ -D -r 2048 -f -b -u $USER -c `nproc``

Además, utilizaremos 3 configuraciones distintas de hardware y kernel para todas las pruebas:

- Laboratorios f-l3109: kernel no-RT (accede por ssh)
- RaspberryPi: kernel no-RT
- RaspberryPi: kernel RT

RBPi Kernel no-RT	**RBPi Kernel RT**
rbpi-01: IP1	rbpi-10: IP6
rbpi-02: IP2	rbpi-11: IP7
rbpi-03: IP3	rbpi-12: IP8
rbpi-04: IP4	rbpi-13: IP9
rbpi-05: IP5	rbpi-14: IP10

Para acceder a las RBPI, que tienen IPs privadas, debes hacerlo siempre desde ordenadores de la universidad. Puedes utilizar tu mismo usuario que en los laboratorios. Además, tendrás tu HOME montado igualmente.

Calcula las latencias máximas y medias en los 3 escenarios descritos arriba (S1, S2, S3) para las 3 configuraciones de hardware y kernel disponibles. Muestra los resultados obtenidos en la siguiente tabla (si lo prefieres puedes usar gráficos o plots de barras mientras la información quede correctamente mostrada).

cyclictestURJC						
	Laboratorios		RaspberryPi			
	Kernel NO RT		Kernel NO RT		Kernel RT	
	Latencia media (ns)	Latencia max (ns)	Latencia media (ns)	Latencia max (ns)	Latencia media (ns)	Latencia max (ns)
S1						
S2						
S3						

2. Desarrollo de cyclictestURJC.

Desarrolla una aplicación en C llamada *cyclictestURJC* que mida la latencia de planificación de un sistema GNU/Linux utilizando las técnicas explicadas en clase. cyclictestURJC debe ejecutar continuamente durante 1 minuto y debe realizar la siguiente funcionalidad:

- Generar un thread por cada core del ordenador. Para saber el número de cores, puedes utilizar la siguiente función:
  - `int N = (int) sysconf(_SC_NPROCESSORS_ONLN);`
- Cada thread debe ejecutarse en la cola de prioridad SCHED_FIFO, con prioridad 99.
  - `pthread_setschedparam`
- Cada thread debe ejecutarse en un core diferente, por tanto, establece la afinidad con
  - `pthread_setaffinity_np`
- El thread debe estar constantemente calculando la latencia de planificación mediante técnicas de sleep. **RECOMENDACIÓN**: realiza esperas del orden de milisegundos.
- Para configurar Linux con la mínima latencia en DMA, asegúrate que tu programa siempre comienza abriendo el fichero `/dev/cpu_dma_latency` y escribiendo un 0. **NO** cierres el descriptor hasta el final del programa (si lo cierras, restablecerá la configuración previa).

```
static int32_t latency_target_value = 0;
latency_target_fd = open("/dev/cpu_dma_latency", O_RDWR);
write(latency_target_fd, &latency_target_value, 4);
```

- Al finalizar el programa (después de 1 minuto de ejecución), debe mostrar el siguiente resumen con la latencia media y latencia máxima por thread/cpu.

```
$ ./cyclictestURJC
[0]    latencia media = 000001984 ns. | max = 000016865 ns
[1]    latencia media = 000001984 ns. | max = 000017124 ns
[2]    latencia media = 000001999 ns. | max = 000014698 ns
[4]    latencia media = 000001995 ns. | max = 000016550 ns
[3]    latencia media = 000002011 ns. | max = 000014936 ns
[5]    latencia media = 000002005 ns. | max = 000014396 ns

Total  latencia media = 000001996 ns. | max = 000017124 ns
```

- Además, tu programa debe guardar en un fichero CSV (cyclictestURJC.csv) en formato texto plano (https://en.wikipedia.org/wiki/Comma-separated_values) todas las medidas obtenidas en el siguiente formato:

CPU, NUMERO_ITERACION, LATENCIA

MUY IMPORTANTE: Asegúrate de realizar esta operación de escritura a fichero, una vez terminada toda la evaluación de la latencia, para no interferir con las interrupciones al escribir.

MUY IMPORTANTE: Asegúrate que cuando realices las mediciones en las RBPi, no haya ningún otro estudiante realizando las mediciones también. Obtendréis resultados erróneos.

Utilizando la herramienta cyclictestURJC que has desarrollado, rellena la siguiente tabla del mismo modo que hiciste en el apartado 1.

cyclictestURJC						
	Laboratorios		RaspberryPi			
	Kernel NO RT		Kernel NO RT		Kernel RT	
	Latencia media (ns)	Latencia max (ns)	Latencia media (ns)	Latencia max (ns)	Latencia media (ns)	Latencia max (ns)
S1						
S2						
S3						

Por último, crea un gráfico que muestre la distribución de la latencia para los distintos escenarios en Raspberry Pi. Deberás incluir los 2 tipos de kernel (real-time y no real-time) y los 3 escenarios definidos (idle, hackbench, etc.). Asegúrate de que todos los gráficos de latencia utilicen el mismo rango en el eje X, expresado en microsegundos, para facilitar la comparación entre ellos. Los datos deben provenir de los archivos CSV generados para cada escenario.

En cada gráfico, que deben aparecer las leyendas correctamente identificando los escenarios, incluye la desviación estándar calculada para cada escenario. Puedes ponerlo en la propia leyenda del plot.

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.hist.html