

# DESARROLLO DE SOFTWARE PARA SISTEMAS EMPOTRADOS

## Práctica 4: Caracterización de la plataforma

Héctor Pérez  
Michael González



2



DSSE: Héctor Pérez

### Informe

Sección 1: Configuración del entorno de desarrollo

Sección 2: Uso de dispositivos de entrada/salida

Sección 3: Uso y configuración de motores

Sección 4: Caracterización de la plataforma

- Objetivos
- Proceso de medida de los parámetros característicos del sistema operativo
- Características del ev3dev para tiempo real


Sección 5: Desarrollo de un sistema empujado

Sección 6: Conclusiones

## Objetivos

- Medidas de parámetros característicos de la plataforma
  - latencia del kernel en la respuesta a eventos
  - atención de interrupciones
  - cambios de contexto


## Introducción

- El **sistema operativo influye** en los tiempos de respuesta del código de usuario
  - gestión de eventos internos y externos
    - *tick* del sistema 
    - interrupciones
  - operaciones adicionales proporcionadas por librerías
  - operaciones ejecutadas a distinta prioridad
    - por ejemplo, servicios del kernel
  - operaciones ejecutadas en exclusión mutua

## Introducción

- Sistema operativo ev3dev
  - Kernel configurado como **CONFIG\_PREEMPT**
    - La mayor parte del código del kernel es expulsable
      - No se espera a que se termine de ejecutar el código del kernel para ejecutar el planificador
      - Excepciones: *secciones críticas del kernel*
    - Utilizado en sistemas con requisitos de latencia en torno a varios milisegundos
  - Todas las operaciones de configuración de los aspectos de tiempo real de una aplicación deben realizarse con **privilegios de administrador**
    - por ejemplo, las políticas y los parámetros de planificación

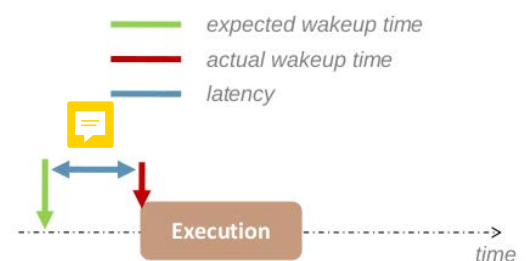
## Latencia en el kernel

- Herramienta *cyclictest*
  - herramienta habitual en sistemas basados en Linux
  - disponible en el paquete **rt-tests** 
  - mide la latencia de respuesta a un evento
    - tiempo entre la generación de la interrupción y la ejecución de la aplicación

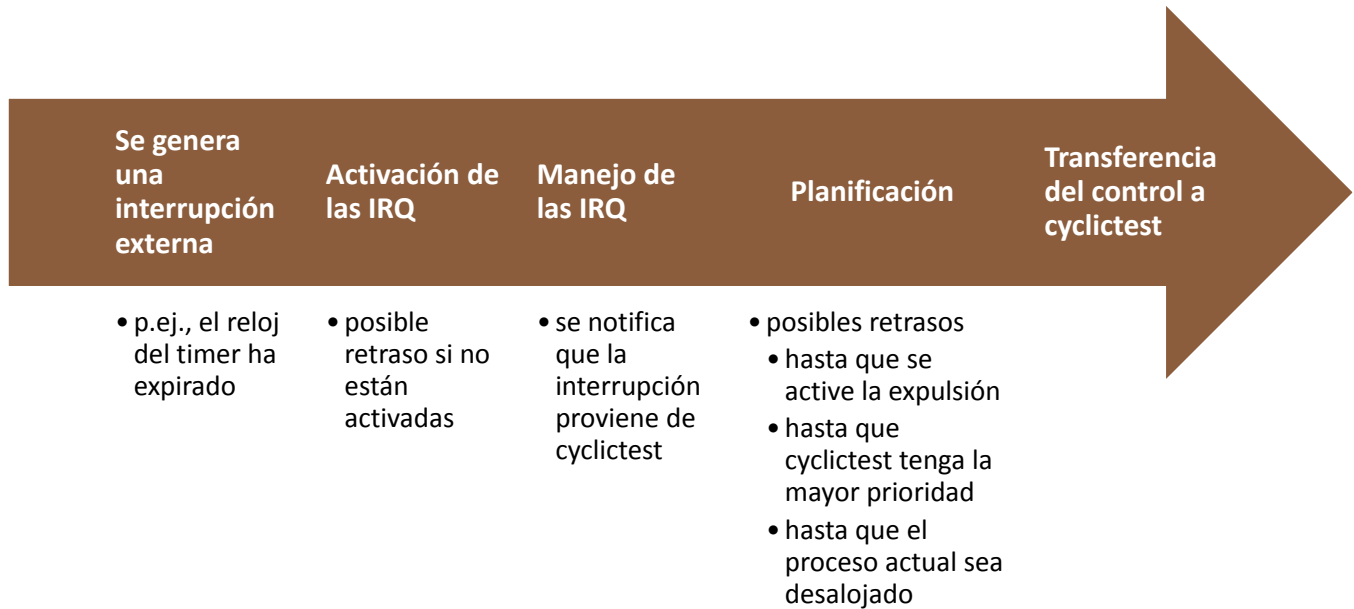
```

clock_gettime(&now))
next = now + par->interval
while (!shutdown) {
    clock_nanosleep(&next)
    clock_gettime(&now)
    diff = calcdiff(now, next)
    ... # update stats
    next += interval
}

```



## Latencia en el kernel



## Latencia en el kernel

- *cyclicttest --help*
  - *-n* *utiliza clock\_nanosleep*
  - *-p NUM* *prioridad del thread*
  - *-h* *genera un histograma en la salida stdout*
  - *-D NUM* *duración del test en segundos*
  - *-m* *desactiva la paginación a memoria SWAP (siempre en RAM)*
  - *-M* *sólo actualiza los resultados al obtener un peor caso*

*Uso: >> cyclicttest -m -M -p 99 -n*
- Ejecutar la herramienta *cyclicttest* en el EV3, primero con prioridad máxima (99) y posteriormente con prioridad 1
  - los resultados temporales están en microsegundos
  - inhabilitar el menú gráfico: *>> systemctl stop brickman*

## Estimación de la latencia en el kernel

- Instala la herramienta **stress-ng** que simula carga en la CPU
  - la herramienta se encuentra en los repositorios oficiales
  - acepta una cantidad elevada de argumentos de entrada. En nuestro contexto, los más relevantes son:
    - sched: política de planificación
    - sched-prio: prioridad
    - cpu-load: porcentaje de carga en la CPU
    - cpu: threads obreros para ejecutar carga de CPU (0 significa el número de cores)
    - cpu-method: algoritmo de carga
    - timeout: duración de la carga para cada algoritmo (segundos)
  - ejemplo:
 

```
stress-ng --timeout 15 --cpu 0 --cpu-load 15
--sched fifo --sched-prio 40 --cpu-method fibonacci
```

## Estimación de la latencia en el kernel

- Ejecutar simultáneamente las aplicaciones **stress-ng** y **cyclicttest** para estimar la latencia de *ev3dev* en los siguientes escenarios
  - razonar los resultados obtenidos

Prioridad de <i>cyclicttest</i>	MAX	MAX	MAX	MAX-1	MAX	MAX
Prioridad de <i>stress-ng</i>	MAX/2	MAX-1	MAX	MAX	MAX/2	MAX/2
Porcentaje de carga	10	10	10	10	40	85

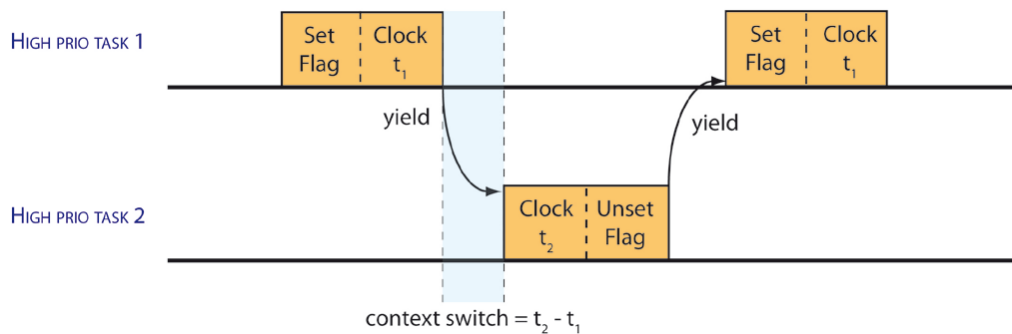
## Estimación de la periodicidad del tick del sistema (1/2)

- Compila la aplicación *tick.c* proporcionada
  - Esta aplicación utiliza la librería *pthread*
  - Explica el algoritmo del código proporcionado e ilústralo mediante una figura
  - Ejecuta la aplicación y razona los resultados obtenidos cuando configuramos un *threshold* de 60 microsegundos
    - relaciona los resultados obtenidos con el flag de configuración del kernel *CONFIG\_HZ*

## Estimación de la periodicidad del tick del sistema (2/2)

- Ejecuta de nuevo la aplicación *tick.c* proporcionada
  - ¿existe algún *overhead adicional* en el sistema?
    - configura un *threshold* apropiado en la aplicación para estimar si dicha sobrecarga tiene alguna periodicidad
    - ¿cómo modelarías ese overhead si quisieras diseñar un sistema de tiempo real?

## Estimación del cambio de contexto: algoritmo



- Las tareas se ejecutan a la misma prioridad
  - al invocar el subprograma *yield* ceden el procesador a otra tarea y se encolan de nuevo en la cola de tareas listas para ejecutar del planificador
- Puede sincronizarse la inicialización mediante un flag
  - dado que la ejecución de las tareas es controlada mediante *yield*, no se utilizarán mecanismos de acceso concurrente seguro por su alto *overhead* en relación con el coste del cambio de contexto

## Estimación del cambio de contexto: desarrollo

- Compilar el fichero *yield.c* proporcionado que implementa el algoritmo descrito anteriormente
  - esta aplicación utiliza la librería *pthread*
  - ejecutar la aplicación y estimar los valores mínimo, máximo y promedio del *coste de cambio de contexto*
  - razonar los resultados relacionándolos con los obtenidos en los apartados anteriores

## Ev3dev, un kernel de Linux de propósito general (1/3)

- Analiza los resultados obtenidos en las tres métricas realizadas (*cyclictest*, *tick* y *yield*)
  - ¿aparecen los mismos *overheads* en cada una de las métricas?
  - piensa en las diferencias entre los tres códigos y el objetivo de un sistema operativo como Linux

## Ev3dev, un kernel de Linux de propósito general (2/3)

- Real-Time Throttling
  - mecanismo del kernel para evitar que alguna tarea de alta prioridad pueda bloquear completamente el sistema
- Parámetros del planificador en el kernel de linux
  - `/proc/sys/kernel/sched_rt_period_us`
    - Tiempo de CPU total para todos los threads del sistema (en  $\mu$ secs)
  - `/proc/sys/kernel/sched_rt_runtime_us`
    - Tiempo de CPU disponible para los threads de tiempo real (en  $\mu$ secs)
- Obtén el valor de ambos parámetros para *ev3dev*
  - relaciónalo con los resultados obtenidos hasta ahora



## Ev3dev, un kernel de Linux de propósito general (3/3)

- Inhabilita esta característica del kernel
  - cambia al usuario *root*: `sudo -s`
  - ejecuta el siguiente comando

```
>> echo "-1" > /proc/sys/kernel/sched_rt_runtime_us
```
  - también puedes editar el fichero con un editor tipo *vi*
- Ejecuta de nuevo las tres métricas realizadas (*cyclictest*, *tick* y *yield*)
  - relaciona los resultados con los obtenidos anteriormente