**Desarrollo de Software para Sistemas Empotrados**

**Cuaderno de Bitácora**

**Julio Escudero Cuesta**

**PRACTICA 1**

Durante las prácticas se va a trabajar con un *dispositivo Lego Mindstorms EV3.* El SO a usar será el EV3Dev, que es un Linux Debian compatible con diferentes plataformas como LEGO MINDSTORMS EV3 y Raspberry Pi. Soporta múltiples lenguajes de scripting y al tener un kernel de Linux, permite utilizar numerosos dispositivos USB y Bluetooth, teclados, joysticks…

**Instalación del sistema operativo ev3dev**

Para instalar este sistema operativo en el Lego, lo más sencillo es bajarse una imagen de EV3Dev desde la [web oficial](https://www.ev3dev.org/). En este caso se utilizará una imagen del kernel proporcionada en Moodle.

Una vez bajada la imagen se pide realizar una copia de la misma con la herramienta *balena-etcher.* Esta herramienta es de código abierto y su principal utilidad es la de escribir archivos de imagen (como .iso y .img) en medios compartidos para así crear tarjetas de memoria SD y unidades flash USB. En este caso se utilizará una tarjeta micro SD para copiar la imagen.

**Configuración de la interfaz de comunicación**

Cuando se termine de copiar la imagen, hay que introducir la tarjeta en el LEGO y arrancarlo. Luego hay que conectar mediante USB el LEGO a un ordenador con Linux y utilizando el comando nm-connection-editor. Este comando permite manejar la conexión a red por parte de NetworkManager, que sirve para simplificar la gestión de redes en ordenadores Linux y otros sistemas operativos Unix.

En mi caso la conexión de red no estaba activada en el ordenador, por lo que hay que activarla directamente a mano, encendiendo el Ethernet en la máquina Linux.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

Tras ejecutar el comando nm-connection-editor se abre una ventana en la que se puede configurar la interfaz de comunicación entre el LEGO y el ordenador en el que se está trabajando. Aquí hay que editar el nombre de la conexión (que puede ser uno cualquiera), introducir la dirección MAC que tenga el LEGO, denegar la conexión de todos los usuarios de la red al LEGO (indicar que solo esté disponible para mi) y seleccionar la conexión compartido en los ajustes para IPV4.

**Configuración del entorno para la compilación nativa**

Para entrar en el dispositivo LEGO a través del ordenador de trabajo lo más sencillo es establecer una conexión ssh. El usuario del LEGO por defecto tiene el nombre de *robot* y la contraseña es *maker*. Al haber establecido la conexión a red un par de pasos atrás el LEGO dispone de una dirección IP por lo que se tiene todo lo necesario para realizar la conexión.

Una vez establecida la conexión hay que instalar el paquete build-essentials. Este paquete contiene una serie de paquetes fundamentales para Debían como son los compiladores gcc y g++ y una serie de librerías de C útiles para el desarrollo.

A continuación se piden una serie de datos:

* Versión del kernel

Con el comando cat /proc/version

Linux ev3dev 4.14.117-ev3dev-2.3.5-ev3 #1 PREEMPT Sat Mar 7 12:54:39 CST 2020 armv5tejl GNU/Linux

* Valor de unos flags de configuración en /boot
  + CONFIG\_PREEMPT: y
  + CONFIG\_HIGH\_RES\_TIMERS: y
  + CONFIG\_HZ: 100

También se pide crea el directorio /home/robot/bin para almacenar los ejecutables y /home/robot/src para almacenar los fuentes.

Lo siguiente es crear un programa “hola mundo” sencillo en el LEGO y compilarlo ahí con la orden gcc -Wall -Werror -o hello hello.c y utilizar la herramienta time (escribir time antes de gcc) para obtener el tiempo de compilación. La opción -Wall habilita todas las alertas de constructores que sean fácilmente evitables o pasables. La opción -Werror transforma todas las alertas en errores.

En mi caso la salida que obtengo es la siguiente

Texto

Descripción generada automáticamente

El valor “real” indica el tiempo transcurrido entre la ejecución de la aplicación y la finalización del proceso.

El valor “user” indica la cantidad de tiempo de CPU gastado en código en modo usuario durante la ejecución del proceso.

El valor “sys” indica la cantidad de tiempo de CPU que transcurre dentro del kernel en la ejecución del proceso.

**Configuración del entorno para la compilación cruzada**

Después se pide instalar un compilador cruzado para ARM9, que es el que utiliza el LEGO LV3 y compilar el programa “hola mundo” utilizando este compilador cruzado. Los binarios están disponibles en el Moodle, junto a una plantilla para editar un Makefile.

Una vez descargado todo, hay que editar el Makefile escribiendo la ruta dónde se tiene descomprimido el compilador cruzado en la variable CROSS\_COMPILER, indicando el nombre del compilador en la variable CC y escribiendo una regla (en mi caso “hello”) que realice la compilación del hello.c

Texto

Descripción generada automáticamente

Se ejecuta el make con la regla “hello” y con la herramienta time y se obtienen los siguientes resultados.

Texto

Descripción generada automáticamente

**Automatización del proceso de desarrollo en *eclipse***

Lo último de esta parte es configurar el IDE eclipse para realizar las prácticas.

Lo primero es instalar una extensión que permita la compilación cruzada en C y C++. Para ello basta con ir a la opción Install New Software dentro del menú Help en Eclipse y buscarla. Luego hay que crear un proyecto C e indicar el compilador cruzado que se va a utilizar y el path a dicho compilador. Después simplemente hay que compilar el programa “hola mundo” que se había hecho previamente, desde Eclipse. Para ello hay que seguir una serie de pasos:

* Establecer el modo de ejecución como aplicación remota en C/C++
* Indicar la aplicación que se va a compilar
* Configurar la ejecución remota en el robot indicando la conexión SSH y diferentes propiedades (Dirección IP, usuario y contraseña)
* Indicar que la conexión va a ser a un host remoto y establecer el path dentro del robot para la compilación de aplicaciones C/C++.

Una vez configurado todo esto, al compilar el programa se puede ver en la terminal de eclipse la salida “Hola mundo” y un mensaje de logout.

Texto

Descripción generada automáticamente

**Configuración del uso de recursos en *eclipse***

Para utilizar las librerías y los archivos de cabecera del EV3 hay que configurar el proyecto para incluirlos (en este caso también aparece la librería pthread que se utilizará más adelante).

*Interfaz de usuario gráfica

Descripción generada automáticamente*

Interfaz de usuario gráfica

Descripción generada automáticamente

Una vez hecho esto ya se puede echar mano de las funciones del EV3

El programa que imprime el mensaje “Hello, World!” en la pantalla del brick es el siguiente:

Texto, Carta

Descripción generada automáticamente

**Depuración remota de aplicaciones en *eclipse***

Para establecer la depuración remota se siguen exactamente los pasos del enunciado.

Primero se copian los ficheros gdbserver y gdb en el directorio /home/robot/bin

Texto

Descripción generada automáticamente con confianza baja

y se escribe la dirección hacia estos dos archivos en las opciones de depuración de eclipse.

**PRACTICA 2**

**Completar el fichero *practica2a.c* para que se compruebe periódicamente si alguno de los botones ha sido pulsado.**

* **Inhabilitar el menú del ev3dev**

Como la tarea a realizar va a ser detectar la pulsación de los botones, en la práctica se recomienda deshabilitar el menú, para que la pulsación de los botones no desemboque en ninguna opción.

Para ello simplemente se realiza una conexión ssh con el robot y se introduce la orden: *sudo systemctl stop brickman*

* **Comprobar periódicamente el pulsado de los botones (periodo = 1 seg)**

Para registrar el pulsado de los botones hay que utilizar el archivo de cabecera lcd\_button.h.

Lo primero de todo es implementar las funciones *ev3\_init\_button() y ev3\_quit\_button()*¸ que inicializan y terminan la disponibilidad de los botones, respectivamente.

La clase *ev3\_button* tiene un enumerado *ev3\_button\_identifier* con 6 valores (uno por cada botón del brick).

La función que mira si un botón ha sido pulsado es *ev3\_button\_pressed()*. Esta función recibe un entero como parámetro y comparándolo los valores del enumerado devuelve si dicho botón ha sido o no pulsado (1 si ha sido pulsado, 0 si no lo ha sido).

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

En la función *is\_button\_pressed()* se realiza la llamada a *ev3\_button\_pressed().*

Texto

Descripción generada automáticamente

Con estas modificaciones en el archivo *practica2a.c* durante la ejecución se detecta cuando se pulsan las teclas (y se mantiene 1 segundo) y en pantalla aparece qué botón se ha pulsado. Si no se pulsa ningún botón aparece un mensaje diciendo que no se han pulsado ningún botón.

* + ¿Qué ocurre si se pulsan dos botones al mismo tiempo?

En ese caso, solo se detecta la pulsación de uno de ellos

* **Comprobar periódicamente el pulsado de los botones (periodo = 10 microsegundos)**

Para cambiar el periodo de medición a 1 microsegundo simplemente hay que cambiar la instrucción *sleep(SLEEP\_DURATION)* por la instrucción *usleep*. Esta instrucción toma el valor pasado por parámetro como microsegundos, en vez de como segundos. Hay que cambiar también el valor de la constante SLEEP\_DURATION a 10 (10 microsegundos). Por último, aumentar el número de iteraciones para que la ejecución sea más larga (cambiar la constante MAX\_ITERATIONS).

* **Razonar las limitaciones del driver para manejar la botonera**

**Ejecutar en un terminal remoto la aplicación button\_stats.**

Hay que copiar el archivo *button\_stats* descargado desde Moodle en el robot.

Interfaz de usuario gráfica

Descripción generada automáticamente

El programa devuelve el código del botón presionado junto al tiempo exacto.

El tiempo mínimo para procesar una pulsación es de unos pocos milisegundos.

**Manejo del sensor de color. Aplicación *practica2b.c***

La aplicación *practica2b.c* carga los sensores del robot y los inicializa. Realiza todas las configuraciones previas necesarias para utilizarlos y en un bucle imprime el color obtenido a través del sensor. Por último libera todos los recursos utilizados.

Para utilizar los leds del brick hay iniciarlos.



También es necesario cambiar el modo de detección del color de ambiente a color.

Texto

Descripción generada automáticamente

Y en el bucle en el que se identifica el color obtenido, indicar que se encienden los leds con el color verde, rojo o amarillo, según corresponda. Al principio se apagan las luces de los leds, para no mezclar colores.

Texto

Descripción generada automáticamente

**Filtrado de datos. Aplicación *practica2c.c***

Para el filtrado de los datos del sensor de color se copia la función proporcionada en el enunciado y se invoca justo después de obtener el dato.

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

Es importante destacar que hay que incluir la librería m en el proyecto.

**Desarrollo de aplicaciones con datos compartidos. Aplicación *practica2d.c***

Se crean 3 hilos. Uno gestiona la pulsación de botones, otro los datos del sensor y el último imprime por pantalla los resultados.

Texto

Descripción generada automáticamente

Los recursos compartidos están protegidos con un mutex.

Texto

Descripción generada automáticamente

Cada hilo ejecuta su parte y acaba cuando se pulsa el botón *BUTTON\_BACK,* que tiene el código 5. Ejemplo del hilo de los botones.

Texto

Descripción generada automáticamente

Al ser un acceso muy corto este en concreto no se ha protegido con el mutex.

Texto

Descripción generada automáticamente

El nuevo hilo reportero queda así:

Texto

Descripción generada automáticamente

Se deja un tiempo el resultado en pantalla y se borra toda la pantalla tras mostrar la

información.

**PRACTICA 3**

**Ejecución aplicación *practica3.c***

Esta aplicación obtiene primero una lista con todos los motores conectados y coge como objetivo el motor conectado al puerto A del robot. A continuación, reinicia el estado del motor al por defecto (recomendado en la documentación justo después de cargar los motores) y prepara el motor para su funcionamiento con la función *ev3\_open\_motor*. Tras esto, procede a la configuración del motor, asignando el modo de parada *COAST,* una velocidad igual al 25% de la velocidad máxima del motor y una posición inicial 0.

A continuación, se entra en un bucle en el que el motor funciona durante un tiempo igual a la iteración en la que se encuentre dentro del bucle multiplicado por 300. En cada iteración se imprime la posición final de la misma y se espera 1 segundo con la función *sleep(1).*

Para terminar, se reinicia la posición del motor, se le ordena que se mueva hasta alcanzar la posición 0 con el comando *RUN\_ABS\_POS*, se reinicia su estado y se borra la lista de motores almacenada.

La orden *usleep()* suspende la ejecución del programa durante un intervalo de microsegundos indicados como parámetro.

La línea *ev3\_motor\_state (belt\_motor) & MOTOR\_RUNNING* comprueba si el motor está funcionando de forma normal. En este caso la línea se encuentra dentro de un bucle while, por lo que se queda esperando hasta que el motor pare (o cambie su estado a otro que no sea de parada, como el *MOTOR\_RAMPING,* que indica que la velocidad está cambiando).

**Aplicación *practica3b.c*. 4 rotaciones del motor**

El fichero *count\_per\_rot* dentro de /sys/class/tacho-motor/motor0 indica que una rotación completa del motor son 360 unidades.

Para hacer que el motor realice 4 rotaciones completas que el motor se active hasta alcanzar la posición 360 multiplicado por la iteración por la que se llegue, empezando por la 1.

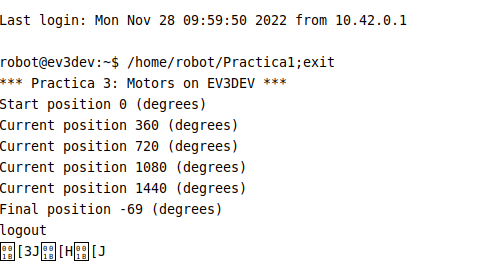
El comando utilizado es el de *RUN\_ABS\_POS*, que indica al motor que se mueva a una posición absoluta. Se ha utilizado este porque cada vez que se realizaba una iteración, el motor se ajustaba a la posición inicial 0, por lo que es necesario decirle que avance desde la posición 0 hasta la 360, 720…

La posición final del motor es de -69 grados.

La posición final del motor justo después de abandonar el estado *MOTOR\_RUNNING* es siempre 0. Si hay un modo de parada que no para el motor en seco, la rotación extra que acumula el motor se cuenta de manera negativa.

Texto

Descripción generada automáticamente

****

**Aplicación *practica3b2.c*. Distintos modos de parada**

Con el modo *COAST,* el motor no se detiene de manera inmediata, lo hace progresivamente. Por ello la posición final del motor era de -69 grados.

Con el modo *BRAKE,* el motor se detiene de manera casi inmediata, aunque sigue manteniendo un leve retraso.

Texto

Descripción generada automáticamente

Con el modo *HOLD,* el motor se detiene de manera inmediata, siendo la posición final de 0 grados.

**Texto

Descripción generada automáticamente**

* **Diferencias al mostrar la posición final del motor**

Como se ha dicho antes, la posición del motor tras salir del estado *MOTOR\_RUNNING* es de 0. Si se muestra la posición tras esperar 1 segundo (con sleep(1)) y si el modo de frenado no es inmediato, la posición del motor varía.

Aquí un ejemplo con el modo de parada *BRAKE*

Texto

Descripción generada automáticamente

**Aplicación *practica3c.c*. Uso de *duty\_cycle.***

Se configura el código para utilizar el modo de funcionamiento *run\_direct y* un *duty\_cycle* de 25.

Texto

Descripción generada automáticamente

Y el resultado es el siguiente

Texto

Descripción generada automáticamente

Como el tiempo que ha estado el motor trabajando son 5 segundos y la posición final es de 849, la velocidad media de giro es aproximadamente 170.

Al aplicar una breve resistencia la posición final es de 810, así que la velocidad de giro decae a aproximadamente 162.

**Aplicación *practica3c2.c*. Uso de *speed.***

Se establece una velocidad de 170 y se cambia el modo de funcionamiento a *run-forever.*

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Y se obtienen obviamente resultados muy similares.

Texto

Descripción generada automáticamente

Además, al incluir una pequeña resistencia, la velocidad de giro también decae como en el apartado anterior.

**Aplicación *cinta.c.* Control preciso del movimiento.**

El diámetro de la rueda enganchada al motor es de 3.7 centímetros. Aplicando la fórmula *L = π·Dw,* la distancia recorrida (L) es igual a *π* multiplicado por el diámetro de la rueda (3.7 cm); 11.6239 aproximadamente.

Con una rotación completa del motor (360º) se recorren 11.6239 cm. La distancia entre la primera rueda de la cinta de montaje y la segunda es de 7.4 centímetros, por lo que para desplazar un objeto desde la primera hasta la segunda se necesita que el motor rote 360\*7.4/11.6239 = 232.28 º aproximadamente.

La distancia entre la segunda rueda y la tercera es de 9.7 cm. Para desplazar un objeto desde la segunda hasta la tercera se necesita que el motor rote 360\*9.7/11.6239 = 298.31 º aproximadamente.

Se introducen las constantes necesarias

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Y el código de configuración de motor y funcionamiento queda así.

Texto

Descripción generada automáticamente

En el bucle se ajusta una posición dependiendo de si el objeto se encuentra en la segunda rueda o en cualquier otra. Una vez ajustada, se avanza hasta esa posición.

Al finalizar se vuelve a la posición inicial y se cierra el motor.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

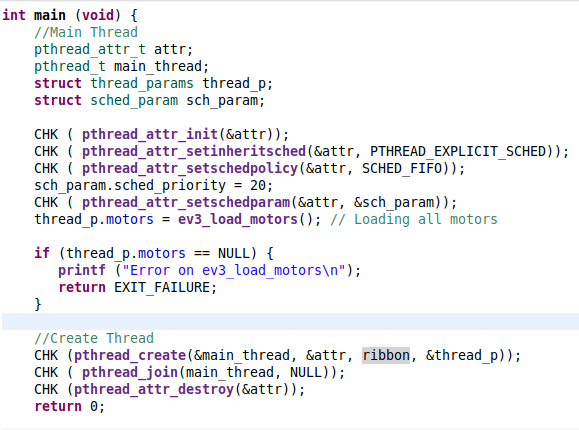
**Aplicación *cinta\_sched.c.* Aplicación planificada por prioridades.**

Para introducir algún atributo al hilo que va a ejecutar la aplicación se le la lista de motores

Imagen que contiene nombre de la empresa

Descripción generada automáticamente

En el main se crea el hilo que va a ejecutar lo que hacía antes la aplicación cinta.c. Se le asigna una política de planificación por prioridades fijas (SCHED\_FIFO) y una prioridad de 20.



La función que ejecuta el hilo es igual al main de la aplicación cinta.c, aunque con pequeñas modificaciones para trabajar con hilos.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Utilizando la herramienta rtinfo\_threads se obtiene lo siguiente,

Texto

Descripción generada automáticamente

donde se aprecia que el proceso ejecutado sigue la política de planificación basada prioridades fijas y una prioridad de 20.

**PRACTICA 4**

**Uso de la herramienta *cyclictest***

Ejecutar la herramienta *cyclictest* en el EV3, primero con prioridad máxima (99) y posteriormente con prioridad 1.

Prioridad 1

Texto

Descripción generada automáticamente

Prioridad 99

Texto

Descripción generada automáticamente

**Ejecución herramienta *stress-ng* y *cyclictest* para estimar la latencia.**

Primera columna:

**Captura de pantalla de computadora

Descripción generada automáticamente**

Segunda columna:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Tercera columna:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Cuarta columna:

Interfaz de usuario gráfica, Aplicación

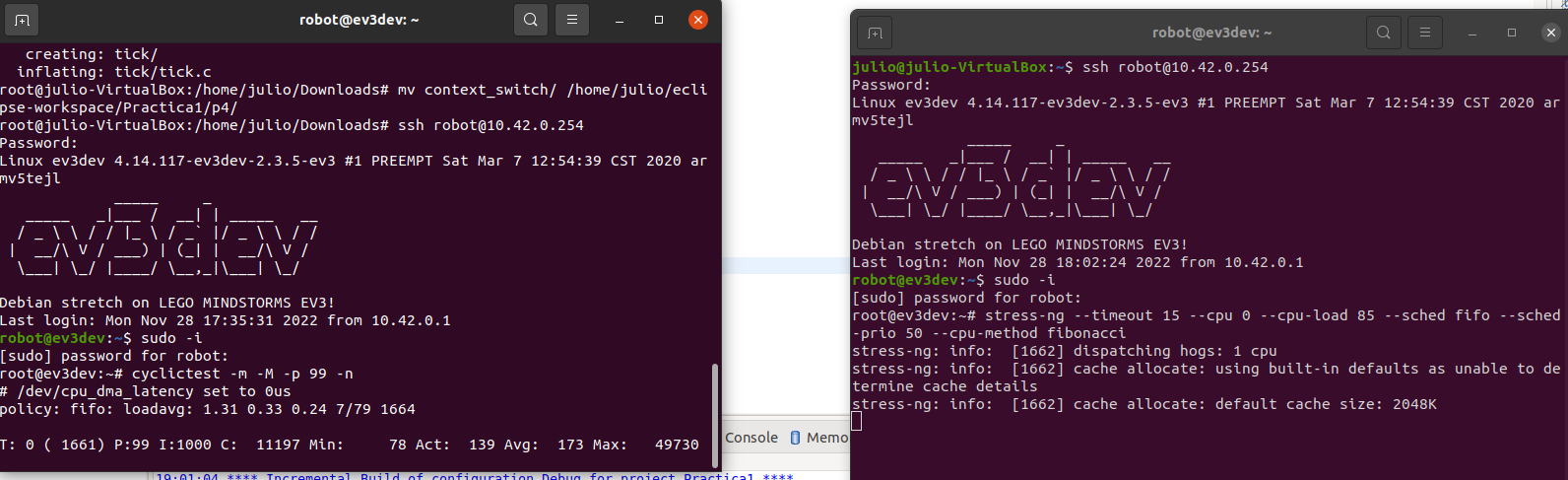
Descripción generada automáticamente

Quinta columna:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Sexta columna:

****

Analizando los resultados se aprecia que la latencia del ev3dev se dispara cuando la herramienta stress-ng simula carga en la CPU con una prioridad máxima. Por otro lado, la latencia también aumenta considerablemente cuando el porcentaje de carga en la CPU sube, aunque dicha carga no se ejecute a una prioridad alta.

**Ejecución aplicación tick.c con umbral de 60 microsegundos.**

**Imagen que contiene parado, agua, grande, tabla

Descripción generada automáticamente**

La aplicación *tick.c* asigna una prioridad igual a la prioridad máxima del sistema - 1 al programa principal y luego configura los atributos de otro hilo, indicándole que es joinable, que utiliza una política de planificación basada en prioridades fijas y que su prioridad de ejecución es la máxima del sistema .

La función que ejecuta el otro hilo creado mida las interrupciones que hacen que el kernel expulse el código de usuario. Aquellas interrupciones que expulsan a la tarea un periodo de tiempo superior al designado en el umbral se imprimen por pantalla, indicando el tiempo de expulsión y cuándo se ha dado la expulsión. Por ello todos los tiempos que aparecen en pantalla son superiores a los 60 microsegundos que se han puesto de umbral.

Gráfico

Descripción generada automáticamente

El flag de configuración del kernel *CONFIG\_HZ* es de 100 microsegundos (visto en la primera práctica). Este flag indica una interrupción periódica del kernel para realizar tareas de mantenimiento. Con él se define el *jiffy* del sistema, que es el tiempo entre dos ciclos de reloj.

La relación entre estas dos variables es 1/HZ = jiffy. Por tanto, en este caso el jiffy es 1/100 = 0.01, es decir, cada 0.01 segundos (10 milisegundos) hay un tick del reloj del sistema.

Esto se aprecia en la imagen de arriba. No hay ningún intervalo entre ticks del reloj superior a los 10 ms, al menos para el umbral establecido de 60 µs.

Por otro lado se aprecian los overheads del sistema. Hay interrupciones que superan los 120 µs que no se corresponden con las interrupciones periódicas del kernel configuradas con el flag *CONFIG\_HZ*. A la hora de diseñar un sistema de tiempo habría que tener en cuenta estas interrupciones cuando se designe los plazos que se pueden cumplir en la realización de las tareas críticas.

**Ejecución aplicación yield.c**

**Texto

Descripción generada automáticamente**

La aplicación yield.c devuelve el tiempo medio, menor y mayor de cambio de contexto registrado al llamar a *sched\_yield().* La llamada a esta función supone que el proceso abandona voluntariamente el procesador sin bloquearse. En este caso se ejecutan dos hilos en paralelo que leen el reloj del sistema y abandonan el procesador para dejarse paso mutuamente.

Se puede ver que el tiempo medio y mínimo son muy bajos, indicando que la mayoría de cambios de contexto se aproximan al tiempo mínimo. Sin embargo el tiempo máximo es de 50079 µs, una medida anómala a las otras dos. Esta medida coincide aproximadamente con la latencia medida con la herramienta cyclictest para el caso de una carga alta en el sistema con una prioridad máxima.

**Parámetros del planificador en el kernel de Linux**

Sched\_rt\_period\_us

Texto

Descripción generada automáticamente con confianza media

Sched\_rt\_runtime\_us

Texto

Descripción generada automáticamente con confianza baja

Con estos datos se aprecia que las tareas de tiempo real pueden utilizar un máximo del 95% de la CPU. Si este 95% de la CPU está ocupado por una tarea de tiempo real, las restantes tareas de tiempo real no podrán ejecutarse, ya que el 5% restantes del sistema está dedicado a tareas que no son de tiempo real.

Esta configuración del sistema Linux tiene como objetivo protegerse de que se bloquee con la ejecución de tareas de tiempo real.

**Ejecuta de nuevo las tres métricas realizadas (*cyclictest, tick* y *yield*)**

Se cambia el valor del *sched\_run\_runtime\_us* a -1.



Cyclictest

**Interfaz de usuario gráfica

Descripción generada automáticamente**

Tick

Imagen que contiene Patrón de fondo

Descripción generada automáticamente

Yield

Texto

Descripción generada automáticamente

Al ajustar el uso máximo de CPU de las tareas de tiempo real a -1, se indica que dichas tareas pueden utilizar el 100% de la CPU. Es por ello por lo que al ejecutar de nuevo las tres métricas (cyclictest, tick y yield) se observa que tanto la latencia máxima del *ev3dev*, como el número de interrupciones del kernel, como el tiempo máximo de cambio de contexto bajan considerablemente.

**PRACTICA 5. TRABAJO FINAL**

**Propósito del proyecto**

El propósito del proyecto es utilizar el robot Lego Mindstorms EV3, con el cuál se ha trabajado en la asignatura, para construir un robot que se desplace, que detecte objetos en frente suyo y que reaccione dependiendo de cómo esté configurado. Podrá atacar con la catapulta o retroceder y buscar otro camino. Además se podrá interactuar con él a través de la pantalla y de los botones del robot.

Todo ello se programará utilizando los conocimientos vistos en clase (planificación con hilos, semáforos, prioridades…).

**Descripción de la arquitectura del sistema**

El sistema está compuesto por 5 hilos y 1 semáforo que controla el acceso a los recursos que comparten dichos hilos. Existe un hilo que controla el funcionamiento de los motores (el movimiento del robot y la catapulta), otro que controla el sensor de infrarrojos, otro que controla el pulsado de los botones del robot, otro que muestra la información en pantalla y otro que determina si el robot está en estado de ataque o no.

**Configuración utilizada**

Todos los hilos se ejecutan continuamente y con la misma prioridad, mientras no se registre una pulsación en el botón BUTTON\_BACK. El único hilo que se ejecuta periódicamente es el que se encarga de refrescar la pantalla con los datos, con un periodo igual a 1 segundo. Esto es debido a que, si este hilo se ejecuta continuamente, la pantalla se actualiza todo el rato y resulta todo un poco borroso. Al principio simplemente se actualizaba la pantalla si había algún nuevo dato, pero esto hacía más complicado el diseño en otras partes, por lo que se optó por la primera opción mencionada.

El hilo que controla los motores tiene como atributos los propios motores que debe controlar. El hilo que controla el sensor también tiene como atributo el propio sensor.

**Algoritmos utilizados**

A continuación se describen los algoritmos utilizados en el código. Se dejan fuera los detalles de el dibujo y escritura en pantalla.

Algoritmo Marchas y cargadoCatapulta

Si ultimo boton pulsado es el boton derecho

poner\_velocida\_alta

Fin Si

Si ultimo\_boton\_pulsado es el boton centro

poner\_velocidad\_media

Fin Si

Si ultimo\_boton\_pulsado es el boton izquierdo

poner\_velocidad\_baja

Fin Si

Si ultimo\_boton\_pulsado es el boton de arriba

Si la catapulta está vacía

Indicar que se va a recargar

FinSi

Fin Si

Fin Si

FinAlgoritmo

Algoritmo Cargando Catapulta

Mientras ultimo boton pulsado no es el boton arriba

Si ultimo boton pulsado es el del centro

Mover gatillo de la catapulta al punto inicial

Indicar que la catapulta está cargada

FinSi

Mirar el último botón pulsado

Fin Mientras

FinAlgoritmo

Algoritmo Objecto delante

Si hay un objeto delante

guardar velocidad actual

parar motores

Si se está en modo ataque y la catapulta está cargada

Disparar

Señalar que la catapulta está vacía

FinSi

Si no

Moverse hacia atrás y hacia la derecha o izquierda

Parar los motores cuando se complete el movimiento

Fin Si

Restaurar velocidad guardada antes de frenar

Fin Si

FinAlgoritmo

Algoritmo Sensor

Mientras el ultimo botón pulsado no sea el de arriba

ver valor del sensor

Si el valor es menor que 100

Señalar que hay un objeto delante

FinSi

Fin Mientras

FinAlgoritmo

Algoritmo Estado

Mientras el ultimo botón pulsado no sea el de arriba

Si la catapulta no está siendo cargada

Si el ultimo botón pulsado es el de abajo

Si el modo ataque está activado

Desactivar el modo ataque

FinSi

Si no

Activar el modo ataque

FinSi

FinSi

FinSi

Fin Mientras

FinAlgoritmo

Algoritmo Botones

Mientras el ultimo botón pulsado no sea el de arriba

Señalar el último botón pulsado

Fin Mientras

FinAlgoritmo

Algoritmo Pantalla

Mostrar el menu principal

Mientras el ultimo botón pulsado no sea el de arriba

Guardar todos los valores de los datos compartidos

Actualizar la pantalla

Esperar el periodo determinado

Fin Mientras

FinAlgoritmo

Algoritmo Actualizar pantalla

Limpiar la pantalla

Mostrar el botón de salida

Mostrar el botón de modo de ataque

Mostrar el botón de estado de la catapulta

FinAlgoritmo

Algoritmo Mostrar Menu Principal

Dibujar botón "Start" relleno

Dibujar botón "Settings" vacío

Mientras no se seleccione ninguna opción

Si se ha pulsado un botón, guardarlo

FinSi

Si el último botón pulsado ha sido el de arriba y la opción "Comenzar" no está seleccionada

Marcarlo como seleccionado y Rellenarlo

Vaciar el botón "Settings"

FinSi

Si el último botón pulsado ha sido el de abajo y la opción "Comenzar" está seleccionada

Desmarcarlo como seleccionado y Vaciarlo

Rellenar el botón "Comenzar"

FinSi

Si el último botón pulsado es el del centro y la opción "Comenzar" está seleccionada

Marcar que se ha seleccionado una opción

Señalar que el robot comienza a funcionar

FinSi

Fin Mientras

FinAlgoritmo

**Instrucciones de utilización**

Cuando se ejecuta el programa aparecen en la pantalla 2 recuadros, uno relleno y otro vacío. El cuadro de arriba representa la opción “Comenzar” y el cuadro de abajo la opción “Configuraciones”. Para seleccionar una opción, hay que pulsar el botón del centro. Por falta de tiempo no se ha implementado nada en la opción “Configuraciones”, aunque si que se puede navegar entre las dos opciones, con los botones de Arriba y Abajo.

Tras seleccionar la opción “Comenzar” el robot se pone en marcha. En la pantalla aparecen varias indicaciones:

* En la esquina superior izquierda aparece la opción **Salir:**

Si se pulsa el botón **Atrás** en el robot, el programa finaliza

* En la parte central de arriba aparece el estado de la catapulta

Por defecto, el programa empieza con la opción **Cargada** (se supone que la catapulta está cargada al inicio del programa). Cuando se disparé, la opción cambiará a **Vacía**.

Si la catapulta está vacía y se pulsa el botón **Arriba,** el robot se detendrá y la opción cambiará a **Cargando.**

Si se pulsa el botón **Central**, el gatillo de la catapulta vuelve a la posición original (desde donde sujeta la catapulta). Tras pulsarlo, la opción cambiará a **Cargada**.

Si se pulsa de nuevo el botón **Arriba**, el robot volverá a moverse.

* En la parte central inferior aparece el modo de ataque del robot

Por defecto, el programa empieza con la opción **Ataque.** Si se detecta algún objeto delante, el robot se detendrá, se disparará la catapulta y el robot volverá a moverse.

Si se pulsa el botón **Abajo**, el modo de ataque se desactiva y la opción cambia a **No ataque.** Si se detecta algún objeto delante, el robot dará marcha atrás y hacia la izquierda o derecha de manera aleatoria y luego continuará funcionando.