DESARROLLO DE SOFTWARE PARA SISTEMAS EMPOTRADOS

Práctica 2: Uso básico de dispositivos de entrada/salida

Héctor Pérez
Michael González

Informe

Sección 1: Configuración del entorno de desarrollo

Sección 2: Uso de dispositivos de entrada/salida

- Objetivos
- Gestión de dispositivos en el kernel
- Uso básico de dispositivos

Sección 3: Uso y configuración de motores

Sección 4: Caracterización de la plataforma

Sección 5: Desarrollo de un sistema empotrado

Sección 6: Conclusiones

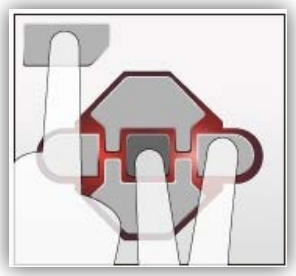
Objetivos

- **Uso básico de dispositivos**
  - desarrollo de aplicaciones en C para manejar dispositivos de entrada/salida sencillos: sensores, botones, LEDs, etc.
    - practicar con el sistema de desarrollo cruzado
    - se utilizarán ejemplos con funcionalidad básica de los que partir en el proyecto personal
  - adquirir experiencia en la lectura de documentación de interfaces de bajo nivel
  - observar las dificultades de interaccionar directamente con dispositivos hardware
- Ejecución de aplicaciones concurrentes POSIX
  - creación y uso de threads o procesos ligeros
- Uso de datos compartidos en entornos POSIX
  - creación y uso de mutexes

Repaso de programación en C

Introducción

- Sistema de desarrollo cruzado (revisar **práctica 1**)
  - Generación de código mediante un IDE (p.ej., *Eclipse*)
  - Compilación cruzada del código fuente
    - integrada dentro del IDE o mediante el terminal
    - utilizar las opciones de compilación y enlazado adecuadas
      - **-Wall -Werror** para mejorar la calidad del código
  - Ejecución remota de aplicaciones en el *brick* de Lego
    - integrada dentro del IDE
    - mediante *ssh* en el terminal
      - recomendable utilizar autenticación mediante llave pública
- Reinicio hardware del *brick* en caso de incidencias
  - aplica la siguiente combinación en la botonera



Introducción

- Gestión de dispositivos en el kernel *ev3dev*

- Control hardware a bajo nivel
- Común a sensores y motores
- Monitorización de eventos

- Habilita la comunicación
- Depende del sensor (I2C, UART, Analógico)

- Gestión de alto nivel del sensor
- Creación de dispositivos virtuales *sensor<N>*

/sys/class/lego-sensor/sensor<N>					
Fichero	*address*	*commands*	*modes*	*value<N>*	*poll_ms*
Uso	Identifica el puerto	Listado de los comandos aceptados	Listado de los modos de operación	Valores Medidos	Periodo de polling

Desarrollo: manejo de la botonera

- Completar el fichero ***practica2a.c*** para que *compruebe periódicamente si* alguno de los botones ha sido pulsado
  - consultar la documentación para el manejo de la botonera en ev3dev
  - puede ser necesario **inhabilitar el menú** del ev3dev
    - sudo systemctl stop brickman
  - mostrar un mensaje por **consola** indicando el botón pulsado o que no se ha pulsado ningún botón
    - ¿qué ocurre si se pulsan dos botones al mismo tiempo?
  - ejecutar la aplicación con distintos periodos
    - utilizar un periodo de 1 segundo e indicar las incidencias detectadas
    - utilizar un periodo de 10 µsegundos e indicar las incidencias detectadas
  - razona las limitaciones del driver disponible para manejar la botonera
- Ejecutar en un terminal remoto la aplicación ***button_stats***
  - estimar el tiempo mínimo que se tarda en procesar una pulsación
  - utilizar el tiempo mínimo calculado como periodo en *practica2a.c*

Dispositivos de entrada/salida: sensores

- Múltiples sensores disponibles en EV3

Dispositivos de entrada/salida: sensores

- Sensor de color EV3

  - obtiene información sobre el entorno
  - 3 modos de funcionamiento:
    - **Color**: reconoce un determinado número de colores
    - **Luz reflejada**: mide la cantidad de luz reflejada por un objeto próximo
    - **Luz ambiente**: mide la intensidad de la luz incidente
  - Posibles usos:
    - resolución del cubo de *Rubic*, detección de objetos, encendido automático, control de brillo, monitorización de plantas, etc.

Desarrollo: manejo del sensor de color

- Realizar el montaje indicado en el fichero *montaje_parte1.pdf*
- La aplicación ***practica2b.c*** muestra cómo se maneja el sensor
  - ejecuta la aplicación y comenta brevemente las etapas del código
  - modifica el código para que se enciendan los LEDs del brick de acuerdo a la lectura del sensor de color
    - configura el sensor para operar en el modo de detección de colores

Color	-	Negro	Azul	Verde	Amarillo	Rojo	Blanco	Marrón
Sensor	0	1	2	3	4	5	6	7

    - únicamente identificaremos los colores ***rojo*** y ***verde*** a través de los LEDs del brick de Lego
      - posteriormente, identificar también el color ***amarillo***
      - explicar el procedimiento utilizado
  - comenta las dificultades encontradas en el uso de este sensor

Algoritmos: filtrado de datos vía software

- Los datos obtenidos por un sensor pueden sufrir alteraciones por ruido
  - (generalmente) fluctuaciones pequeñas y contínuas
- Se pueden aplicar diferentes técnicas de filtrado
  - p.ej., un filtro paso bajo permite configurar la respuesta del sistema ante los cambios
    - la constante ***ALPHA*** determina la capacidad de filtrado y la capacidad de respuesta del sistema a los cambios

```
#define ALPHA   0.3

float lowpass_filter (float new_data, float old_data) {
    static bool first_time = true;
    if (first_time) {
        old_data = new_data;
        first_time = false;
    }

    float result = old_data + (ALPHA * (new_data - old_data));
    return result;
}
```

Desarrollo: filtrado de los datos de luz ambiente

- Crear un fichero ***practica2c.c*** que filtre los datos que proporciona el sensor de color
  - configura el sensor para operar en el modo de detección de luz ambiente
    - el sensor proporciona un valor entero en el rango [0 , 100]
- Muestra los resultados de los datos obtenidos y los datos filtrados a través de la consola o la pantalla LCD
  - puede ser recomendable redondear el valor filtrado para mostrarlo por pantalla
- La aplicación termina cuando se pulsa el botón *BUTTON_BACK*

Anexo: desarrollo de software en entornos POSIX

- Aplicaciones concurrentes
  - Creación de threads en sistemas operativos *POSIX*

```
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine)(void *),
                   void *arg);
```

Parámetro	Significado
thread	identificador del thread creado
attr	define los atributos del thread
start_routine	puntero a la función a ejecutar por el thread
arg	argumento que se pasa a la función a ejecutar
Retorno	**Cero si se ha ejecutado con éxito** **Cualquier otro valor en caso de error**

Anexo: desarrollo de software en entornos POSIX

- Uso de datos compartidos
  - Inicialización de mutexes en sistemas operativos *POSIX*

```
pthread_mutex_t mutex;
int pthread_mutex_init
      (pthread_mutex_t *mutex,
       const pthread_mutexattr_t *attr);
```

Parámetro	Significado
mutex	mutex a inicializar
attr	define los atributos del mutex (*protocolo de sincronización, techo de prioridad, inter-proceso*)
Retorno	**Cero si se ha ejecutado con éxito** **Cualquier otro valor en caso de error**

Anexo: desarrollo de aplicaciones con datos compartidos

- Crea un fichero ***practica2d.c*** a partir de los códigos *practica2b.c* y *practica2c.c* que sea concurrente y maneje datos compartidos. Por ejemplo:
  - el thread principal identifica los botones pulsados
    - la aplicación termina cuando se pulsa el botón *BUTTON_BACK*
    - una vez pulsado, el *thread principal* queda a la espera de que el resto de los threads de la aplicación terminen sus ejecuciones
    - muestra por consola un mensaje de finalización
  - un segundo thread se encarga de leer los datos del sensor de color y de filtrarlos

Anexo: desarrollo de aplicaciones con datos compartidos

- La aplicación contiene un tercer thread *reportero* que se encarga de mostrar **toda** la información de los dispositivos
  - en concreto, escribe por la pantalla *LCD* la siguiente información:
    - el botón pulsado o que no se ha pulsado ningún botón
    - el valor obtenido con el sensor de color
    - el modo de funcionamiento del sensor de color
  - puede resultar útil la función **sprintf**
  - los otros threads **no** muestran información por pantalla/consola
  - *Nota*: al utilizar el LCD, es recomendable inhabilitar el menú del *ev3dev*
- La escritura / lectura de datos mostrados por la LCD están protegidos mediante un ***mutex***
  - asegura la consistencia de datos en cada iteración
- Indica en el informe los periodos de activación de cada thread