Universidad Autónoma de Baja California

Facultad de Ciencias Químicas e Ingeniería

SISTEMAS EMBEBIDOS

Práctica 2: Tasks

Docente: Evangelina Lara Camacho

Alumno: Gómez Cárdenas Emmanuel Alberto

Matricula: 01261509

Objetivo

El alumno se familiarizará con el uso de las tareas usando el sistema embebido ESP32 DevKit v1 para desarrollar aplicaciones para sistemas basados en microcontrolador para aplicarlos en la resolución de problemas de cómputo, de una manera eficaz y responsable.

Equipo

Computadora personal con conexión a internet.

Teoría

- **Describa a detalle la función *xTaskCreatePinnedToCore* para crear una tarea que este anclada a un core en particular. Incluya un ejemplo de uso.**

La función ***xTaskCreatePinnedToCore*** es utilizada para crear una tarea y fijarla a un núcleo en particular, esto puede ser útil para optimizar el rendimiento y gestionar eficientemente las tareas.

Sintaxis y Parámetros:

```
BaseType_t xTaskCreatePinnedToCore(TaskFunction_t pvTaskCode, const char *const
pcName, const uint32_t usStackDepth, void *pvParameters, UBaseType_t uxPriority,
TaskHandle_t *pxCreatedTask, const BaseType_t xCoreID);
```

- **Valor de Retorno:** La función regresa ***pdPASS*** si la tarea fue creada con éxito, o un error definido en el archivo ***projdefs.h***
- **pvTaskCode:** Puntero a la función de entrada.
- **pcName:** Nombre descriptivo de la tarea.
- **usStackDepth:** Tamaño del stack de la tarea especificado por número de bytes.
- **pvParameters:** Puntero a los parámetros dados a la tarea.
- **uxPriority:** La prioridad a la que debe correrse la tarea.
- **pvCreatedTask:** Usado para pasar el ***handler*** por el cual se referenciará la tarea.
- **xCoreID:** Núcleo en el cual se atará la tarea (***tskNo_AFFINITY***, para indicar que no se atará a ningúno en particular).
- **uxMemoryCaps:** Memory Capabilites de la memoria de la pila de la tarea

Ejemplo de Uso

Un ejemplo podría ser un sistema de procesamiento de señales de audio, donde el procesamiento del audio se asigna a un núcleo, mientras que todas las demás tareas, como la interfaz de usuario, la lectura de sensores y la conectividad, se ejecutan en el otro núcleo. Esto permite que el procesamiento de audio, que es muy exigente y debe realizarse en tiempo real, se ejecute sin interrupciones, asegurando un rendimiento óptimo sin interferencias de tareas menos críticas.

- **Describa a detalle la función *xEventGroupSync* la cual activa bits dentro de un grupo de eventos y luego espera a que se active una combinación de bits dentro del mismo grupo de eventos. Incluya un ejemplo de uso diferente al de la documentación.**

La función ***xEventGroupSync*** es utilizada para sincronizar múltiples tareas en un sistema de tiempo real mediante la activación y espera de ciertos bits dentro de un grupo de eventos. Esto es útil cuando varias tareas necesitan alcanzar un estado común antes de continuar su ejecución, logrando una coordinación precisa.

Sintaxis y Parámetros:

```
EventBits_t xEventGroupSync(EventGroupHandle_t xEventGroup, const EventBits_t uxBitsToSet, const EventBits_t uxBitsToWaitFor, TickType_t xTicksToWait);
```

- **Valor de Retorno:** La función devuelve los bits del grupo de eventos al desbloquear la tarea. Si se alcanzan los bits esperados, se desbloquea. Si el tiempo de espera expira, puede desbloquearse sin que todos los bits esperados hayan sido activados.
- **xEventGroup:** Es el identificador del grupo de eventos que está siendo utilizado para sincronizar.
- **uxBitsToSet:** Especifica los bits que la tarea debe activar dentro del grupo de eventos cuando llama a la función.
- **uxBitsToWaitFor:** Especifica los bits que la tarea debe esperar a que estén activados antes de continuar. La tarea se bloquea hasta que todos estos bits estén activos.
- **xTicksToWait:** Especifica el tiempo que la tarea esperará antes de que ocurra un timeout si los bits esperados no se activan. Un valor de portMAX_DELAY indicaría que la tarea esperará indefinidamente.

Ejemplo de Uso

Un ejemplo de uso sería un escenario de procesamiento paralelo donde múltiples tareas cargan partes de un archivo desde diferentes fuentes (por ejemplo, de la memoria interna, red, y almacenamiento externo), ninguna de las tareas puede continuar hasta que todas hayan completado su parte de la carga.

Desarrollo

Implemente en el ESP32 ESP-IDF el juego pong haciendo el uso de **tareas y mecanismos de sincronización de tarea**. La implementación debe ser eficiente en el uso de recursos de computo (procesador, memoria y periféricos).

Pong es un juego con temática de tenis de mesa. El juego consiste en un apala y una pelota. El jugador controla la pala y debe evitar que la pelota caiga. La pelota rebota al impactar la pala. El juego termina cuando el jugador no acierta con la pelota y la pelota cae o cuando el jugador decide terminar.

La pelota se lanza automáticamente a la pantalla. El jugador debe mover la pala a la izquierda o a la derecha para alinearla con la pelota. Cuando hagan contacto, la pelota rebota y se mueve hacia el otro lado de la pantalla. La pelota u la pala no deben salir de los limites de la pantalla. Cuando la pelota llega a un límite, la pelota rebota. Cuando la pala llega a un límite, no puede avanzar mas hacia ese lado. Cada vez que la pala toca la pelta, se incrementa en 1 la puntuación del jugador.

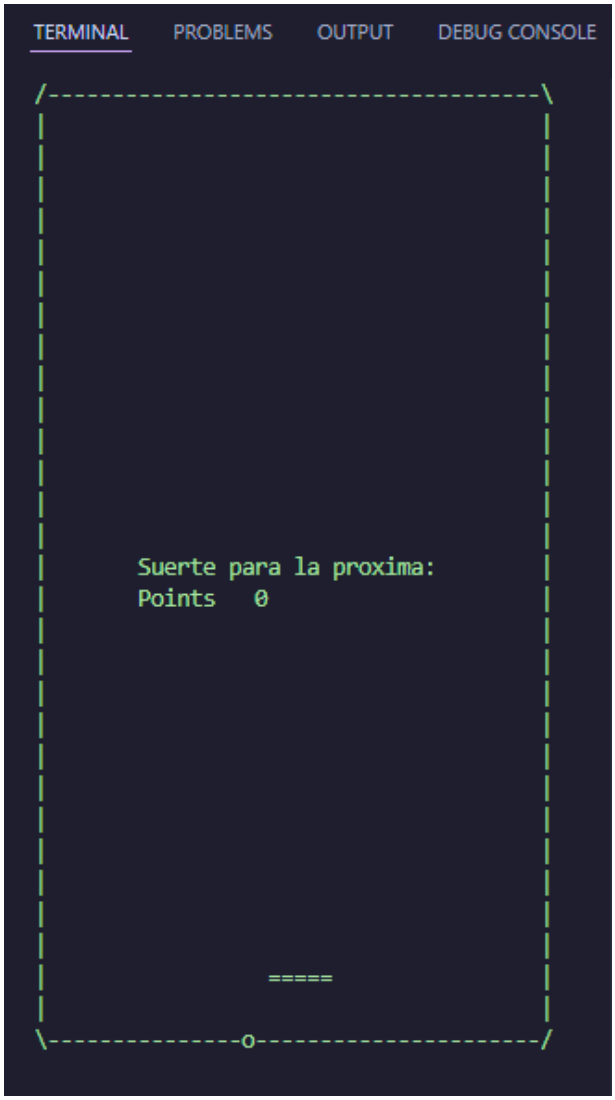
Al final del juego, si el jugador decidió terminarlo, se despliega el mensaje, hasta “Hasta luego”. O si el jugador perdió, se despliega “Suerte para la próxima”.

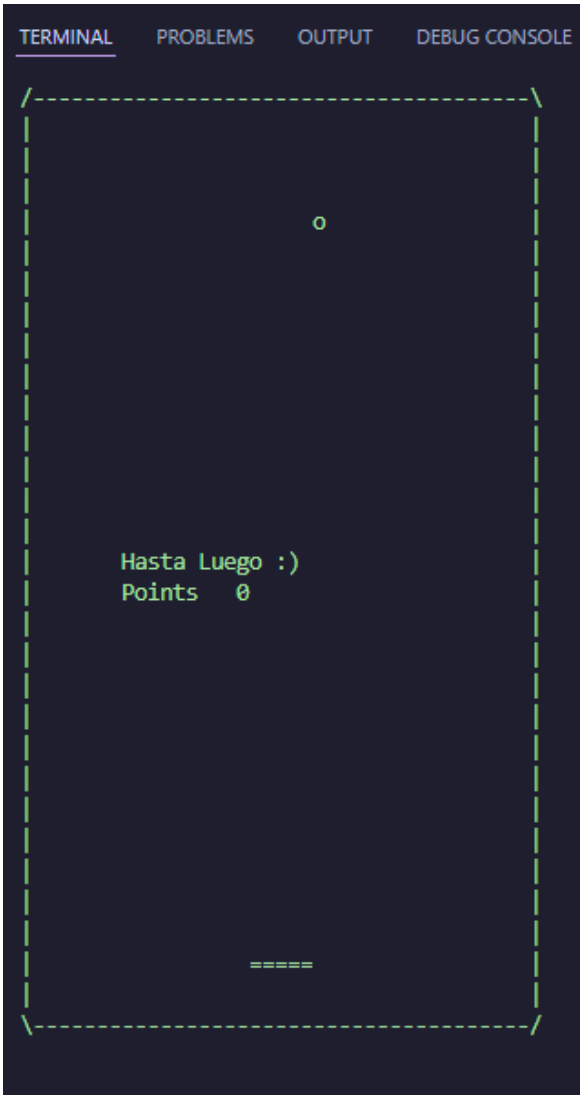
El tamaño de la terminal puede ser manipulado mediante las definiciones CONSOLE_HEIGHT y CONSOLE_WIDTH del archivo pong.h. En este ejemplo se usaron los valore 30 y 40 respectivamente.

A continuación, se presentan los dos casos.

Imagen 1: Suerte para la próxima (El usuario ha perdido)

Imagen 2: Hasta Luego (El usuario ha terminado el juego)





Conclusiones y Comentarios

El uso de las tareas y especialmente el poder anclar una tarea a un núcleo en especifico hace de la optimización un proceso más sencillo al poder enfocarte en un par de funciones para un núcleo y lo restante al otro, lo cual lo veo muy interesante.

Dificultades en el Desarrollo

El uso de los grupos de sincronización fue algo que no le di mucha importancia, sin embargo, después de utilizarlos me di cuenta de que si tienen una gran utilidad.

Referencias

https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/index.html

Código

El código fuente puede ser encontrado en el [Repositorio de GitHub “Sistemas Embebidos”]

Al ser múltiples archivos, solo presentaré archivo principal “main.c”

```c
#include "console.h"
#include "esp_log.h"
#include "esp_random.h"
#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "freertos/queue.h"
#include "freertos/task.h"
#include "pong.h"

#define TAG "Tasks"
uint64_t points = 0;
QueueHandle_t buttonQueueHandler;
EventGroupHandle_t xGameEventSync;
int32_t lastStateChange = 0;
uint32_t currentTime = 0;
game_elements_t gameElements = {
    .paddle = {.y = PADDLE_FIXED_Y,
               .x = PADDLE_INITIAL_X,
               .last_x = 1,
               .size = PADDLE_SIZE,
               .speed = PADDLE_SPEED,
               .symbol = PADDLE_SYMBOL},

    .ball = {.x = BALL_INITIAL_X,
             .y = BALL_INITIAL_Y,
             .last_x = 1,
             .last_y = 1,
             .symbol = BALL_SYMBOL,
             .vertical_direction = MOVING_UP,
             .horizontal_direction = MOVING_LEFT},
};

void delayMillis(int millis) { vTaskDelay(pdMS_TO_TICKS(millis)); }
```

```
void configGPIOs() {
   gpio_config_t io_conf;

   io_conf.intr_type = GPIO_INTR_NEGEDGE;
   io_conf.mode = GPIO_MODE_INPUT;
   io_conf.pin_bit_mask = (1 << BUTTON_LEFT) | (1 << BUTTON_END) |
                          (1 << BUTTON_RIGHT) ;
   io_conf.pull_down_en = 1;
   io_conf.pull_up_en = 0;
   gpio_config(&io_conf);

   io_conf.intr_type = GPIO_INTR_DISABLE;
   io_conf.mode = GPIO_MODE_OUTPUT;
   io_conf.pin_bit_mask =
       (1 << LED_DEBUG) | (1 << LED_1S) | (1 << LED_2S) |
       (1 << LED_4S)    | (1 << LED_8S) | (1 << LED_16S);
   io_conf.pull_down_en = 0;
   io_conf.pull_up_en = 0;
   gpio_config(&io_conf);
}

static void IRAM_ATTR buttonInterruptHandler(void *args) {
   uint32_t buttonActioned = (uint32_t)args;
   currentTime = xTaskGetTickCount() * portTICK_PERIOD_MS;
   if (currentTime - lastStateChange < BUTTON_BOUNCE_TIME) {
      return;
   }
   if (buttonActioned == BUTTON_LEFT) {
      gpio_set_level(LED_DEBUG, 1);
   } else if (buttonActioned == BUTTON_RIGHT) {
      gpio_set_level(LED_DEBUG, 0);
   }
   lastStateChange = currentTime;
   xQueueSendFromISR(buttonQueueHandler, &buttonActioned, NULL);
}
```

```
void movePaddle(void *args) {
   int pinNumber;
   while (true) {
      if (xQueueReceive(buttonQueueHandler, &pinNumber, portMAX_DELAY)) {
         if (pinNumber == BUTTON_LEFT) {
            if (gameElements.paddle.x > 0 + BORDER_WIDTH +
                gameElements.paddle.speed) {
               gameElements.paddle.x -= gameElements.paddle.speed;
            } else {
               gameElements.paddle.x = 0 + BORDER_WIDTH;
            }
         } else if (pinNumber == BUTTON_RIGHT) {
            if (gameElements.paddle.x <
                CONSOLE_WIDTH - gameElements.paddle.size - BORDER_WIDTH -
                gameElements.paddle.speed) {
               gameElements.paddle.x += gameElements.paddle.speed;
            } else {
               gameElements.paddle.x = CONSOLE_WIDTH - gameElements.paddle.size;
            }
         } else
            xEventGroupSetBits(xGameEventSync, GAME_ENDED);
      }
      delayMillis(MINIMUM_DELAY_MS);
   }
}

void configInterruptions() {
   buttonQueueHandler = xQueueCreate(10, sizeof(uint32_t));
   xTaskCreate(movePaddle, "Move Paddle Task", 2048, NULL, 1, NULL);

   gpio_install_isr_service(0);
   gpio_isr_handler_add(BUTTON_LEFT,  buttonInterruptHandler,
                        (void *)BUTTON_LEFT);
   gpio_isr_handler_add(BUTTON_END,   buttonInterruptHandler,
                        (void *)BUTTON_END);
   gpio_isr_handler_add(BUTTON_RIGHT, buttonInterruptHandler,
                        (void *)BUTTON_RIGHT);
}

void startGame() {
   xTaskCreatePinnedToCore((TaskFunction_t)renderGameTask,
                           "Render the Game", 2048, &gameElements, 1, NULL, 0);
   xTaskCreate((TaskFunction_t)displayPointsTask,
               "Show Score", 2048, &points, 1, NULL);
}
```

```
void app_main() {
   configGPIOs();
   initializeUart();
   configInterruptions();
   gameElements.ball.y = esp_random() % (CONSOLE_HEIGHT / 2) + 1;
   gameElements.ball.x = esp_random() % CONSOLE_WIDTH;
   startGame();
   xGameEventSync = xEventGroupCreate();
   xEventGroupWaitBits(xGameEventSync, ALL_EVENTS, pdTRUE, pdTRUE,
portMAX_DELAY);
}
```