Universidad de los Andes Colombia | Departamento de Ingeniería Eléctrica y Electrónica

Sebastián Guerrero Ríos – 202021249
Nicolás Rincón Sánchez – 202021963

IELE3222 – ARQUITECTURA Y DISEÑO DE SISTEMAS DIGITALES

DESIGNPROBLEM-M03 (Tamagotchi)

Descripción del dispositivo

El Tamagotchi consiste en un dispositivo alimentado a través de una ESP32, que contiene una mascota virtual que interactúa con el usuario. La tarjeta de desarrollo ESP32 hace uso del sistema operativo FreeRTOS. De esta manera, se emplearon estructuras de datos para garantizar la concurrencia en el sistema.

El dispositivo cuenta adicionalmente con conexión a un acelerómetro y a una pantalla OLED. Así mismo, cuenta con tres botones que permiten que el usuario interactúe con la interfaz principal y las capacidades de la mascota. En la pantalla se muestra en todo momento el estado actual del juego, donde el centro lo ocupa la mascota animada. Los botones permiten interactuar con las opciones y seleccionar alguna para que afecte el estado de la mascota.

Diseño e implementación de la concurrencia

Se definieron los siguientes requerimientos y la manera de implementarlos con las estructuras de FreeRTOS (colas y semáforos):

Menú y selección de opciones

- Para esto se generó una tarea que corre de manera continua cuando se genera una interrupción con el botón central.
- Se implementó una máquina de estados que controla la opción en la cual se encuentra el menú en un momento, la selección de dicha opción y el desplazamiento con los botones laterales entre las opciones.
- La máquina de estados también controla el tipo de mensaje que se envía cuando se oprime el botón central y a qué cola se dirige. Las demás tareas se implementan con colas, de forma que, cuando una cola se llena con un elemento, la tarea recibe el valor de la cola para que ocurra un cambio de contexto y la tarea en cuestión inicie el quantum.

Creación de barras de estadísticas

- Se implementan mediante la tarea *vEstadisticas* que aprovecha el estado actual de la máquina de estados. De esta manera, cuando el sistema se encuentra en un estado diferente al que muestra las opciones del menú, aparecen dibujadas las barras en la parte superior izquierda.
- Esta tarea se ejecuta constantemente, de manera que las barras van incrementando o decrementando a medida que cambian los valores de las estadísticas.

- Esta tarea no interactúa con el mútex puesto que nunca debe modificar los datos compartidos, sino sólo leerlos.

Tarea de Casa

- Se implementa mediante la cola *casaQueue* que recibe un mensaje con un 1 para mostrar el bitmap del escenario y a la mascota ubicada en el centro. Sin embargo, cuando se oprimen los botones laterales, se envía un 2 o un 3 dependiendo del botón oprimido. En estos casos, dependiendo del valor enviado a la cola, se incrementa o decrementa la posición horizontal. Así, se vuelve a dibujar el escenario y se coloca a la mascota en una posición diferente más a la derecha o a la izquierda del centro. Esta tarea no interactúa con el mutex pues nunca accede a los datos compartidos.

Estado general de la mascota

- Se implementa una clase mascota con atributos de estado numéricos inicializados en valores intermedios (10, con excepción de la edad que inicia en 0).
- Se implementa el semáforo *xDataMutex* el cual se emplea para bloquear el acceso a los atributos de la mascota siempre que ocurre un proceso de actualización.
- Se tiene un contador de tiempo de 20 segundos que reduce las métricas de la mascota como son felicidad y limpieza, toda vez que incrementa las de hambre y sueño.
- El mútex se utiliza cada vez que otra tarea actualiza los campos de la mascota para bloquear el estado de la actualización y que sólo dicha tarea pueda actualizar.

Tarea de Alimentación

- Cuando el usuario selecciona la opción de “Comer”, se envía una señal de alimentación a una cola llamada *comidaQueue* que tiene un tamaño de 4 y un tamaño de elemento de 4 Bytes (pues almacena enteros).
- La tarea de alimentación verifica si hay elementos en la cola de alimentación. En tal caso, toma el control del sistema.
- El usuario puede emplear las interrupciones con los botones laterales para elegir el tipo de comida que le quiere suministrar a la mascota.
- Al alimentar a la mascota, el mútex de actualización de estado toma los recursos del sistema y aumenta la métrica para simular que el tamagotchi ha comido, incrementando la salud (sólo en el caso que haya consumido la manzana, es decir, que se haya enviado a la cola el mensaje de interrupción del botón derecho) y la satisfacción del hambre. Cuando finaliza la actualización, se liberan los recursos. Esto se hace para evitar un bloqueo entre la disminución automática de las métricas y la actualización de la tarea de alimentación.

Tarea de Juego

- Cuando el usuario selecciona la opción de “Jugar”, se envía una señal de juego a la cola llamada *juegoQueue* que tiene tamaño 1.
- A continuación, la tarea de juego toma el control del sistema. El juego consiste en un Piedra-Papel-Tijeras, en el cual el usuario emplea el botón izquierdo para desplazarse por el menú de opciones. Una vez selecciona una opción, se interrumpe el espacio de selección y se da paso para que el oponente elija automáticamente su opción.
- Esto se logra mediante el envío de los mensajes a la cola. Cuando se oprime el botón de desplazamiento entre las opciones, se envía un 2 a la cola, lo cual se traduce en un movimiento entre el menú interno del juego. Cuando se oprime el botón de selección, se envía un 3 a la cola y el sistema genera una opción de manera aleatoria basado en el mismo orden del menú interno del juego. Se realiza la comparación entre la opción de la

máquina y la opción del usuario según las reglas del juego y se determina si ha sido victorioso, ha resultado en empate o ha perdido.
- Se muestra en pantalla un mensaje del resultado del juego (ya sea que haya ganado, perdido o empatado). Finalmente, se emplea el semáforo *xDataMutex* para bloquear los recursos y actualizar el valor de felicidad.

Tarea de Lavado

- Esta tarea se sincroniza con otra llamada *vMovimientoTask*, la cual se comunica con el acelerómetro y recibe sus valores de aceleración en los tres ejes. Cuando se encuentra en ejecución, mide constantemente los valores reportados por el sensor y envía mensajes a la cola *lavadoQueue.*
- Constantemente envía un entero 1, el cual es recibido por la tarea *vWashTask* para pintar la interfaz de lavado. Sin embargo, cuando detecta un valor de la aceleración mayor a 11 en los tres ejes según el reporte del acelerómetro, envía 2.
- Cuando en la cola se envía el 2, la tarea *vWashTask* realiza la petición de recursos con el mútex *xDataMutex*, que bloquea las estadísticas para que se actualice el valor de la higiene y no haya conflictos con la actualización automática de las estadísticas.

Tarea de Dormir

- En esta tarea, se aplica un principio similar de envío de mensajes a la cola *dormirQueue*, pues cuando recibe un valor de 1, pinta el escenario de dormir. Sin embargo, sólo al oprimir el botón izquierdo y generarse la interrupción, se envía un valor de 2 y se cambia el dibujo para mostrar a la mascota durmiendo.
- La tarea de dormir se sincroniza con la tarea de actualización mientras duerme *vUpdateDormirTask* por medio de un entero “dormir”. Cuando se interrumpe con el botón izquierdo, se cambia el valor a 1, con lo cual se piden los recursos compartidos con el mútex *xDataMutex* y se incrementa el valor de la energía. Cuando se interrumpe con el botón derecho, se cambia el valor a 0, con lo cual se liberan los recursos para que nuevamente las estadísticas se actualicen automáticamente mientras la mascota está despierta.

Tarea de Matar

- Esta tarea se ejecuta constantemente y se encarga de revisar que ninguna de las estadísticas de la mascota se encuentre en cero. En el momento en que cualquiera llegue a cero, bloquea los recursos para cambiar el estado de la máquina de estados a 7 y muestra la pantalla de Game Over de manera permanente hasta que se reinicie la ESP.

Diagrama de Tareas, Colas y Semáforos