Carrera de Especialización en Sistemas Embebidos
Sistemas Operativos en Tiempo Real 2.

Autor: Alejandro Celery
Fecha de creación: 04/09/2018

Práctica de memoria dinámica:

El objetivo de esta práctica es aprender a trabajar con algoritmos de asignación de memoria dinámica aptos para su uso en sistemas de tiempo real.
Estos algoritmos están orientados a tener un tiempo de ejecución corto y determinista. Al privilegiar se deja en segundo plano el uso óptimo de la memoria.

No se pretende que el alumno escriba un asignador de memoria dado que hay numerosas implementaciones disponibles en la red, sino que se entrega un gestor de pools de memoria para que el alumno use y que se puede analizar en detalle.

Para la misma se requiere una instalación de Python 3, dado que se usará un script para enviar datos a la EDU-CIAA por UART.

Descripción conceptual de la práctica:

Se propone implementar un servidor de procesamiento de texto. El mismo recibe los datos por UART usando la interrupción de “receptor no vacío” y verifica que conformen al protocolo descrito más abajo. Luego de validar los datos recibidos se los somete a distintos procesamientos según el campo “Operación” del protocolo y se los envía de vuelta por el mismo canal.
La aplicación transmitirá por la misma UART la memoria disponible en el stack de cada tarea que se cree y el heap disponible al iniciar la aplicación.
Para el envío de paquetes de texto a la aplicación se usará un script en python que leerá un archivo de texto y enviará una a una todas las líneas del mismo. Los caracteres que no sean texto no se modificarán. Para una primera aproximación no es importante preocuparse por caracteres del tipo ASCII-extendido (acentos, ñ, etc).

Arquitectura de la aplicación:

Se adopta para la práctica una arquitectura con una tarea y dos colas (una de entrada y una de salida) por cada operación a realizar. Este esquema es representativo de una conexión ethernet, donde suele haber varios procesos distintos transando datos sobre una misma conexión y se usa un port para identificar a cada uno.
Para la memoria dinámica se usarán N pools de memoria de tamaño fijo. La cantidad de pools y el tamaño de los mismos se dimensionarán según la memoria disponible en el microcontrolador usado buscando maximizar el aprovechamiento de la misma.

Paquetes de datos:

1B	1B	1B	T BYTES	1B
STX	OP	T	DATOS	ETX

Delimitación de paquete:

STX: Carácter 0x55
ETX: Carácter 0xAA.

Campos:

OP (Operación):
0: Convertir los datos recibidos a mayúsculas.
1: Convertir los datos recibidos a minúsculas.
2: Reportar stack disponible.
3: Reportar heap disponible.
4: Mensajes de estado de la aplicación.

T (Tamaño): El tamaño del texto recibido.
DATOS: Texto a procesar. Deben ser caracteres ASCII legibles.

Requerimientos de software para la aplicación de la práctica:

R1: La aplicación procesará los paquetes de datos recibidos según el protocolo descrito anteriormente.
R1.1: La lectura de los datos se hará mediante la interrupción de “dato recibido por UART”.
R1.2: Se ignorarán los paquetes que no empiecen con STX.
R1.3: Se ignorarán los paquetes que no contengan ETX al final de los datos.
R1.4: Los paquetes válidos se copiarán a memoria dinámica asignada según el tamaño del paquete.
R1.5: Se pondrá un puntero a paquete válido con operación 0 en la cola “queMayusculizar”.
R1.6: Se convertirán a mayúsculas los paquetes recibidos en la cola “queMayusculizar”.
R1.7: Se pondrá un puntero a paquete válido con operación 1 en la cola “queMinusculizar”.
R1.8: Se convertirán a minúsculas los paquetes recibidos en la cola “queMinusculizar”.
R1.9: Solo se modificarán los caracteres alfabéticos (a-z, A-Z).

R2: La aplicación devolverá por el mismo canal los paquete procesados según el protocolo descrito anteriormente.
R2.1: Se pondrá un puntero a paquete mayusculizado en la cola “queMayusculizados”.
R2.2: Los paquetes recibidos en la cola “queMayusculizados” se transmitirán por la misma UART.
R2.3: Se pondrá un puntero a paquete minusculizado en la cola “queMinusculizados”.
R2.4: Los paquetes recibidos en la cola “queMinusculizados” se transmitirán por la misma UART.
R2.5: El envío de los datos se hará mediante la interrupción de “transmisor vacío”.
R2.6: Al terminar de enviar un paquete se liberará la memoria dinámica asignada para el mismo.

R3: Se usará un algoritmo de asignación de memoria con tiempo de ejecución determinista.
R4: Se usará un algoritmo de asignación de memoria que no produzca fragmentación del sistema.

Requerimientos opcionales:

R5: Luego de cada procesamiento de datos se reportará por UART el mínimo de stack disponible en la tarea afectada, con el campo “Operación” en su valor correspondiente.
R6: Al iniciar la aplicación se reportará por UART el heap disponible con la operación correspondiente, con el campo “Operación” en su valor correspondiente.

Sugerencias:

- Implementar los requisitos de a uno y verificar el funcionamiento de los mismos antes de pasar al siguiente.
- Resolver primero las mayúsculas y luego las minúsculas.
- Implementar el procesamiento de datos con una FSM.
- Dejar para el final la optimización del uso de la memoria.

Historial de cambios

2018/09/04 - FB: Correcciones de formato.