```c
/*
 * General.c
 *
 *  Created on: May 24, 2019
 *      Author: julian
 */

#include "General.h"


/*=============================================================================
                        Almacena en el buffer de la RX ISR
=============================================================================*/
void Add_IncommingFrame(UBaseType_t uxSavedInterruptStatus ,BaseType_t xHigherPriorityTaskWoken,
volatile char c){
    char *PtrSOF = NULL;
    char *PtrEOF = NULL;
    void* XPointerQueUe = NULL; /*Puntero auxiliar  a cola*/

    /*Verifica Inicio de trama*/
    if( Frame_parameters._SOF == c) Data.StartFrame = 1;

    if(Data.StartFrame){
        /*Proteger acceso al buffer*/
        uxSavedInterruptStatus = taskENTER_CRITICAL_FROM_ISR();
        Data.Buffer[Data.Index++]= c;
        taskEXIT_CRITICAL_FROM_ISR(uxSavedInterruptStatus);
    }
    else return;

    if(Data.Index > sizeof(Data)-1) Data.Index =0;  /*Garantiza no desbordamiento del buffer*/

    Data.Buffer[Data.Index] = 0;                    /*char NULL pos siguiente*/

    if(Frame_parameters._EOF == c){
        Data.StartFrame = 0;
        Data.Ready = 1;
        /*Frame buena en el buffer*/

        xTaskNotifyFromISR(xTaskHandle_RxNotify,0,eNoAction,&xHigherPriorityTaskWoken);
        Data.Index =0;
    }
}


/*=============================================================================
                        selecionar puntero a cola segun operacion
 =============================================================================*/
void* SelecQueueFromOperation(Enum_Op_t OP){
    void * XpointerSelected = NULL;
    switch(OP){

    case OP0:   /*Operacion 0*/
        XpointerSelected = xPointerQueue_OP0;
        break;
    case OP1:   /*Operacion 1*/
        XpointerSelected = xPointerQueue_OP1;
        break;
    case OP2:   /*Operacion 2*/
        XpointerSelected = xPointerQueue_OP2;
        break;
    case OP3:   /*Operacion 3*/
        XpointerSelected = xPointerQueue_OP3;
        break;
    }
```

```
    return XpointerSelected;
}

/*=====================================================================================
                                    packetToLower
 ====================================================================================*/
void packetToLower(uint8_t *ptrToPacketLower){

    uint16_t tSizePacket;
    uint8_t i;
    tSizePacket = ((*(ptrToPacketLower + OFFSET_TAMANO)) -'0')*10;
    tSizePacket = tSizePacket + ( (*(ptrToPacketLower+OFFSET_OP+OFFSET_TAMANO)) -'0');
    for(i = 0; i < tSizePacket ; i++){
        if( *(ptrToPacketLower + i + OFFSET_DATO) >= MIN_LOWER &&  *(ptrToPacketLower + i +
        OFFSET_DATO) <= MAX_LOWER)
            *(ptrToPacketLower + i + OFFSET_DATO) = *(ptrToPacketLower + i + OFFSET_DATO) +
            UP_LW_LW_UP;
    }
}
/*=====================================================================================
                                    packetToUpper
 ====================================================================================*/
void packetToUpper(uint8_t *ptrToPacketUpper){
    uint16_t tSizePacket;
    uint8_t i;
    tSizePacket = ( *( ptrToPacketUpper + OFFSET_TAMANO) -'0')*10;
    tSizePacket = tSizePacket + ( *( ptrToPacketUpper + OFFSET_OP+OFFSET_TAMANO) -'0');
    for(i = 0;i < tSizePacket; i++){
        if( *(ptrToPacketUpper + i + OFFSET_DATO) >= MIN_UPPER &&  *(ptrToPacketUpper + i +
        OFFSET_DATO) <= MAX_UPPER)
            *(ptrToPacketUpper + i + OFFSET_DATO) = *(ptrToPacketUpper + i + OFFSET_DATO)-UP_LW_LW_UP;
    }
}
/*=====================================================================================
                                    Print string buffer + message con mutex
 ====================================================================================*/
void PrintUartBuffMutex(char * Message,char *Buf, SemaphoreHandle_t SemMutexUart){
    xSemaphoreTake(SemMutexUart,portMAX_DELAY);
    printf(Message,Buf );
    xSemaphoreGive(SemMutexUart);
}
/*=====================================================================================
                                    Print only message con mutex
 ====================================================================================*/
void PrintUartMessageMutex(char * Message, SemaphoreHandle_t SemMutexUart){
    xSemaphoreTake(SemMutexUart,portMAX_DELAY);
    printf(Message);
    xSemaphoreGive(SemMutexUart);
}
/*=====================================================================================
                                    task create
 ====================================================================================*/
void TaskCreateAll(void){

    xTaskCreate(TaskTxUart, (const char *)"TaskTxUart",configMINIMAL_STACK_SIZE*2, NULL,
    tskIDLE_PRIORITY + 1, NULL);
    xTaskCreate(TaskService, (const char *)"TaskService",configMINIMAL_STACK_SIZE*2, NULL,
    tskIDLE_PRIORITY + 2, &xTaskHandle_RxNotify);
    xTaskCreate(Task_ToMayusculas_OP0, (const char *)"Task_ToMayusculas_OP0",configMINIMAL_STACK_SIZE
    *2, NULL, tskIDLE_PRIORITY + 1, NULL);
    xTaskCreate(Task_ToMinusculas_OP1, (const char *)"Task_ToMinusculas_OP1",configMINIMAL_STACK_SIZE
    *2, NULL, tskIDLE_PRIORITY + 1, NULL);
    xTaskCreate(Task_ReportStack_OP2, (const char *)"Task_ToMayusculas_OP2",configMINIMAL_STACK_SIZE*
    2, NULL, tskIDLE_PRIORITY + 1, NULL);
    xTaskCreate(Task_ReportHeap_OP3, (const char *)"Task_ToMinusculas_OP3",configMINIMAL_STACK_SIZE*2
```

```
    , NULL, tskIDLE_PRIORITY + 1, NULL);
}

/*=====================================================================================
                                queue create
 ====================================================================================*/
void QueueCreateAll(void){

    xPointerQueue_OP0   = xQueueCreate(16 , sizeof(char *)); /*Create queue OP0*/
    xPointerQueue_OP1   = xQueueCreate(16 , sizeof(char *)); /*Create queue OP0*/
    xPointerQueue_OP2   = xQueueCreate(16 , sizeof(char *)); /*Create queue OP0*/
    xPointerQueue_OP3   = xQueueCreate(16 , sizeof(char *)); /*Create queue OP0*/
    xPointerQueue_3     = xQueueCreate(16 , sizeof(char *)); /*Create queue OP0*/

}
/*=====================================================================================
                                semaphore create
 ====================================================================================*/

void semaphoreCreateAll(void){
    SemTxUart    =  xSemaphoreCreateBinary();
    SemMutexUart =  xSemaphoreCreateMutex() ;
}
/*=====================================================================================
                                    conversions
 ====================================================================================*/

char* itoa(int value, char* result, int base) {
    // check that the base if valid
    if (base < 2 || base > 36) { *result = '\0'; return result; }

    char* ptr = result, *ptr1 = result, tmp_char;
    int tmp_value;

    do {
        tmp_value = value;
        value /= base;
        *ptr++ = "zyxwvutsrqponmlkjihgfedcba9876543210123456789abcdefghijklmnopqrstuvwxyz" [35 + (
        tmp_value - value * base)];
    } while ( value );

    // Apply negative sign
    if (tmp_value < 0) *ptr++ = '-';
    *ptr-- = '\0';
    while(ptr1 < ptr) {
        tmp_char = *ptr;
        *ptr--= *ptr1;
        *ptr1++ = tmp_char;
    }
    return result;
}
/*=====================================================================================
                                    Servicio
 ====================================================================================*/

void Service(Module_Data_t *obj ){

    char *PtrSOF = NULL;
    char *PtrEOF = NULL;
    void* XPointerQueUe = NULL; /*Puntero auxiliar  a cola*/
    char *PcStringToSend;

    PcStringToSend = NULL;

    /*Proteger datos para hacer copia local*/
```

```
    taskENTER_CRITICAL();
    Frame_parameters.BufferAux = obj->pvPortMallocFunction(sizeof(Data.Buffer));
    strcpy((char*)Frame_parameters.BufferAux,(const char*)Data.Buffer);
    taskEXIT_CRITICAL();

    /*Buscar posición del inicio de la trama*/
    PtrSOF = strchr((const char*)Frame_parameters.BufferAux, Frame_parameters._SOF);

    if( PtrSOF != NULL ){
        /** Decodificar T :  T[0] -'0' *10 + T[1] - '0'*/
        Frame_parameters.T[0] =  ( *(PtrSOF +  OFFSET_TAMANO)-'0' )*10 + (*(PtrSOF +  OFFSET_TAMANO +
         1)-'0' ) ;

        /** Decodificar OP */
        Frame_parameters.Operation = *(PtrSOF +  OFFSET_OP)-'0';

        /* Cantidad de memoria a reservar*/
        obj->xMaxStringLength = Frame_parameters.T[0] + NUM_ELEMENTOS_REST_FRAME;
    }

    /*Selecionar operaacion*/
    XPointerQueUe = SelecQueueFromOperation(Frame_parameters.Operation);

    if(XPointerQueUe != NULL){
        if (PcStringToSend == NULL) PcStringToSend = obj->pvPortMallocFunction( obj->xMaxStringLength
         );
        /*Envía el puntero al buffer con la trama a la cola*/
        ModuleDinamicMemory_send2(obj,PcStringToSend,0,NULL,(char*)Frame_parameters.BufferAux,
        XPointerQueUe ,portMAX_DELAY);
    }
    /*Libero memoria del buffer aux*/
    ModuleData.vPortFreeFunction(Frame_parameters.BufferAux );
}
/*=====================================================================================
                                    Report  Heap = 1 or stack = 0
 ======================================================================================*/

void Report( Module_Data_t *obj , char * XpointerQueue, uint8_t SelectHeapOrStack){

    char *BSend;
    uint64_t Heap_Stack;
    char BuffA[20];
    char * PcStringToSend = NULL;


    PcStringToSend = NULL;
    BSend = ModuleDinamicMemory_receive(obj,XpointerQueue,  portMAX_DELAY);
    Heap_Stack = SelectHeapOrStack ? xPortGetFreeHeapSize() : uxTaskGetStackHighWaterMark(NULL);

    itoa(Heap_Stack ,BuffA,10);

    /*Puntero donde se copia el stack*/
    if (PcStringToSend == NULL) PcStringToSend = obj->pvPortMallocFunction(strlen(BuffA)+
    NUM_ELEMENTOS_REST_FRAME);
    if(PcStringToSend != NULL){
        sprintf(PcStringToSend+2,"%02d%s}",strlen(BuffA),BuffA);
        *PcStringToSend = *BSend;
        *(PcStringToSend + 1) = *(BSend+1);
    }

    // Enviar a cola de TaskTxUARt
    ModuleDinamicMemory_send2(obj,PcStringToSend,0,NULL,PcStringToSend, xPointerQueue_3,portMAX_DELAY
    );

    /*Libera memoria dinamica {300} recibido del buffer*/
```

```
    ModuleDinamicMemory_Free(obj, BSend);


}
```