DESIGN & IMPLEMENT A SMALL OS DESIGN

Youssef Ahmed Abbas Mohamed

SPRINTS

Sprints

Contents

Project Introduction:

The SOS system is a small operating system designed for time-triggered task scheduling on microcontrollers. It provides a priority-based preemptive scheduler that allows the execution of tasks at specified intervals. The system supports task creation, modification, and deletion, as well as the ability to start and stop task execution dynamically.

The SOS system consists of the following main components:

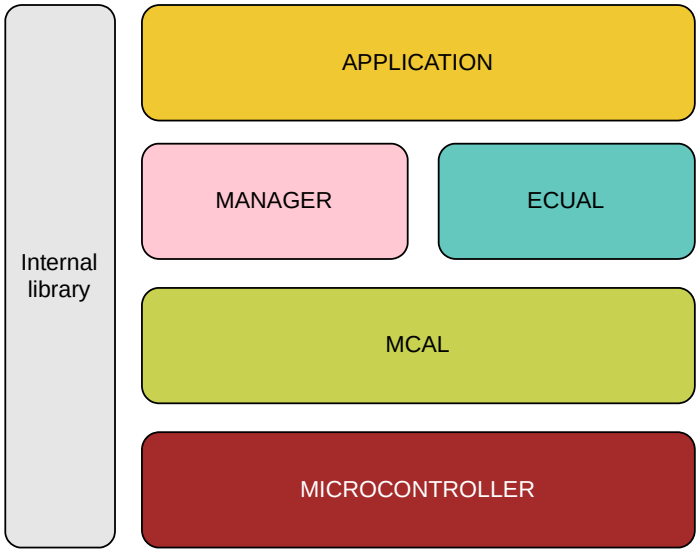
- Main Program: The entry point of the system, responsible for initializing the SOS, configuring peripherals, and managing other program tasks.
- SOS Module: The core module that handles task management, scheduling, and execution.
- Task Functions: User-defined functions that perform specific actions at specified intervals.
- Interrupt Service Routines (ISRs): Interrupt handlers that respond to button presses and control the execution of the SOS.

The system follows a time-triggered approach, where tasks are executed periodically based on their assigned intervals. The SOS module manages the task list, handles task execution, and ensures proper prioritization of tasks.

High Level Design:

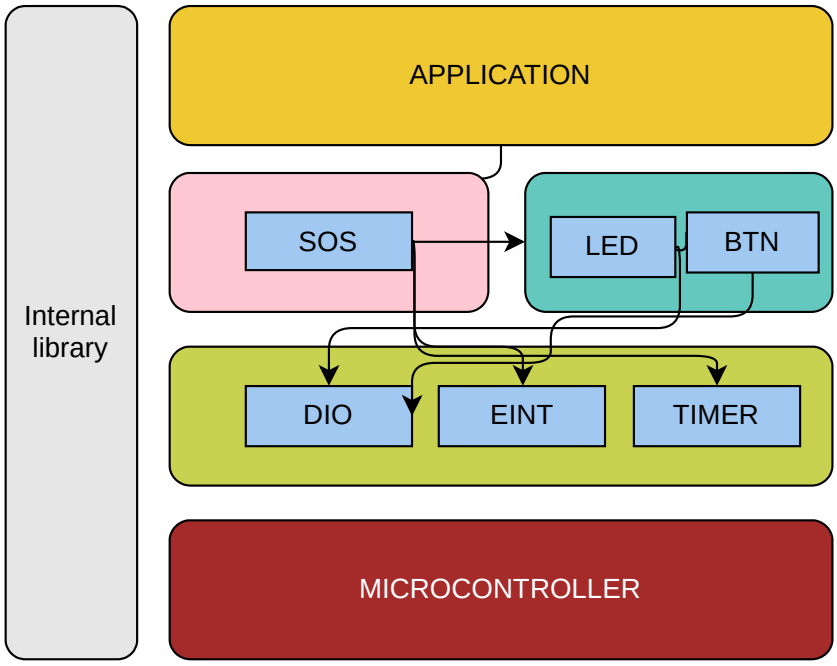
Layered architecture:

1. Application
2. Manager
3. ECUAL
4. MCAL
5. Microcontroller



Module Description

1. Application
2. Manager
    a. SOS
3. ECUAL
    a. LED
    b. Button
4. MCAL
    a. Dio
    b. Timer
    c. EINT
5. Microcontroller



LED:

The module contains functions for initializing the LED, turning it on and off, and toggling its state.

To use this module, the **dio_interface.h** header file must be included. Additionally, the **str_dio_t** structure is used to configure the underlying digital input/output (DIO) pins associated with the LED.

Dependencies

- **dio_interface.h**: This header file defines the functions and data structures related to digital input/output (DIO) operations.

Data Types

enm_led_status_t

This enumerated type defines the possible states of the LED. It has the following values:

- **LED_ON**: Represents the LED being turned on (value: 1).
- **LED_OFF**: Represents the LED being turned off (value: 0).

str_led_t

This structure represents the LED and its associated properties. It contains the following members:

- **str_dio**: An instance of the **str_dio_t** structure that configures the DIO pins associated with the LED.
- **enm_led_status**: The current status of the LED, which can be either **LED_ON** or **LED_OFF**.

Functions

void LED_init(str_led_t* led)

This function initializes the LED by configuring the DIO pins and setting the initial LED status.

- **led**: A pointer to the **str_led_t** structure representing the LED to be initialized.

void LED_on(str_led_t* led)

This function turns the LED on by setting the appropriate DIO pin(s) to the active state.

- **led**: A pointer to the **str_led_t** structure representing the LED to be turned on.

void LED_off(str_led_t* led)

This function turns the LED off by setting the appropriate DIO pin(s) to the inactive state.

- **led**: A pointer to the **str_led_t** structure representing the LED to be turned off.

void LED_toggle(str_led_t* led)

This function toggles the state of the LED. If the LED is currently on, it will be turned off, and vice versa.

- **led**: A pointer to the **str_led_t** structure representing the LED to be toggled.

SOS:

sos_init function:

- Initialize the SOS database.
- Reset any necessary variables or hardware to their default state.
- Configure the timer for time-triggered scheduling.
- Enable global interrupts.

sos_deinit function:

- Disable the SOS.
- Reset any variables or hardware to their default state.
- Clear the task list and task count.

sos_create_task function:

- Check if the maximum number of tasks has been reached. If yes, return an error.
- Create a new task with the provided interval, task function, and priority.
- Add the task to the task list.
- Increment the task count.
- Return success.

sos_delete_task function:

- Iterate through the task list to find the task with the specified task function.
- If the task is found, remove it from the task list and adjust the task count.
- Return success.
- If the task is not found, return an error.

sos_modify_task function:

- Iterate through the task list to find the task with the specified task function.
- If the task is found, update its interval and priority with the provided values.
- Return success.
- If the task is not found, return an error.

sos_run function:

- Check if the SOS is already running. If yes, return.
- Set the SOS running flag to true.
- Enter a loop to execute tasks.
- Iterate through the task list based on task priority.
- For each task, increment its counter.
- If the counter exceeds the task's interval, reset the counter and execute the task's function.
- Repeat the loop until the SOS running flag is false.

sos_disable function:

- Set the SOS running flag to false.

DIO:

Documentation: DIO (Digital Input/Output) Interface

Overview

DIO (Digital Input/Output) interface, which facilitates controlling and reading digital signals on specific pins and ports. The module includes functions for initializing pins, writing values to pins and ports, reading values from pins and ports, and toggling pin states.

To use this module, the **std_types.h** and **dio_private.h** header files must be included. The module defines enums for ports, pin values, pin directions, and DIO errors. It also includes a structure **str_dio_t** for representing a DIO pin.

Dependencies

- **std_types.h**: This header file provides standard types used throughout the module.
- **dio_private.h**: This header file provides private definitions and declarations for the DIO module.

Enums

enm_dio_port_t

This enumerated type defines the available ports for DIO pins. It has the following values:

- **PORT_A**: Represents Port A.
- **PORT_B**: Represents Port B.
- **PORT_C**: Represents Port C.
- **PORT_D**: Represents Port D.

enm_dio_value_t

This enumerated type defines the possible values for a DIO pin. It has the following values:

- **DIO_LOW**: Represents a low logic level (value: 0).
- **DIO_HIGH**: Represents a high logic level.

enm_dio_dir_t

This enumerated type defines the possible directions for a DIO pin. It has the following values:

- **DIO_IN**: Represents the input direction (value: 0).
- **DIO_OUT**: Represents the output direction.

enm_dio_error_t

This enumerated type defines the possible errors that can occur during DIO operations. It has the following values:

- **DIO_FAIL**: Represents a failure or error (value: 0).
- **DIO_SUCCESS**: Represents a successful operation.

Structures

str_dio_t

This structure represents a DIO pin and its associated properties. It contains the following members:

- **port**: The port to which the pin belongs (**enm_dio_port_t**).

- **pin**: The number of the pin within the port.

Functions

enm_dio_error_t dio_init(str_dio_t dio_pin, enm_dio_dir_t dir)

This function initializes a DIO pin with the specified direction.

- **dio_pin**: The **str_dio_t** structure representing the DIO pin to be initialized.
- **dir**: The desired direction for the pin (**DIO_IN** or **DIO_OUT**).

enm_dio_error_t dio_write_pin(str_dio_t dio_pin, enm_dio_value_t value)

This function writes a value to a DIO pin.

- **dio_pin**: The **str_dio_t** structure representing the DIO pin to be written to.
- **value**: The value to be written to the pin (**DIO_LOW** or **DIO_HIGH**).

enm_dio_error_t dio_toggle(str_dio_t dio_pin)

This function toggles the state of a DIO pin. If the pin is currently high, it will be set to low, and vice versa.

- **dio_pin**: The **str_dio_t** structure representing the DIO pin to be toggled.

enm_dio_error_t dio_read_pin(str_dio_t dio_pin, uint8 *value)

This function reads the value of a DIO pin and stores it in the provided variable.

- **dio_pin**: The **str_dio_t** structure representing the DIO pin to be read.
- **value**: A pointer to a variable where the pin value will be stored (**DIO_LOW** or **DIO_HIGH**).

enm_dio_error_t dio_write_port(enm_dio_port_t port, enm_dio_value_t value)

This function writes a value to the specified DIO port. The value will be applied to all pins of the port.

- **port**: The port to which the value will be written (**PORT_A**, **PORT_B**, **PORT_C**, or **PORT_D**).
- **value**: The value to be written to the port (**DIO_LOW** or **DIO_HIGH**).

enm_dio_error_t dio_read_port(enm_dio_port_t port, uint8 *data)

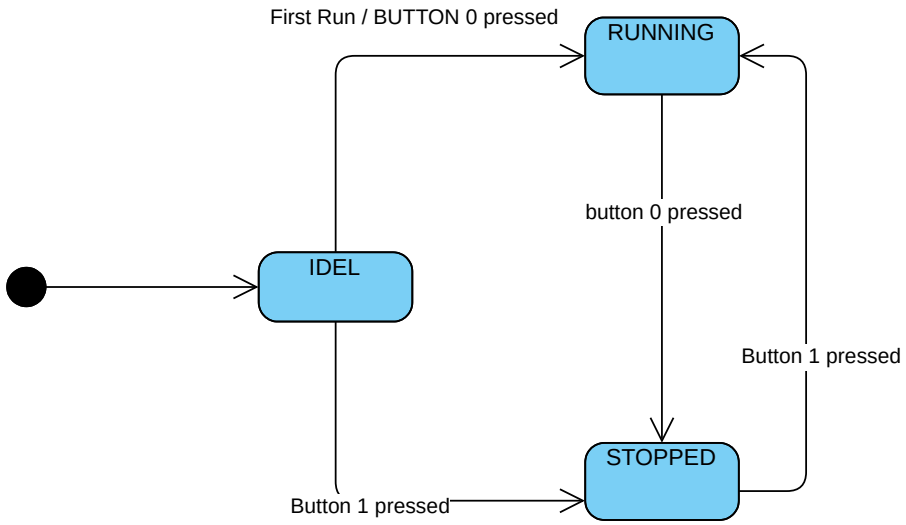
This function reads the value of a DIO port and stores it in the provided variable. The value represents the combined state of all pins in the port.

- **port**: The port to be read (**PORT_A**, **PORT_B**, **PORT_C**, or **PORT_D**).

- **data**: A pointer to a variable where the port value will be stored.

UML:

State Machine:



The state machine for the application consists of three states:

State: IDLE

Description: The system is idle and waiting for a button press.

Events:

- PBUTTON0_PRESSED: Transition to the STOPPED state.

- PBUTTON1_PRESSED: Transition to the RUNNING state.

State: RUNNING

Description: The system is running and executing tasks.

Events:

- PBUTTON0_PRESSED: Transition to the STOPPED state.

- PBUTTON1_PRESSED: No action.

State: STOPPED

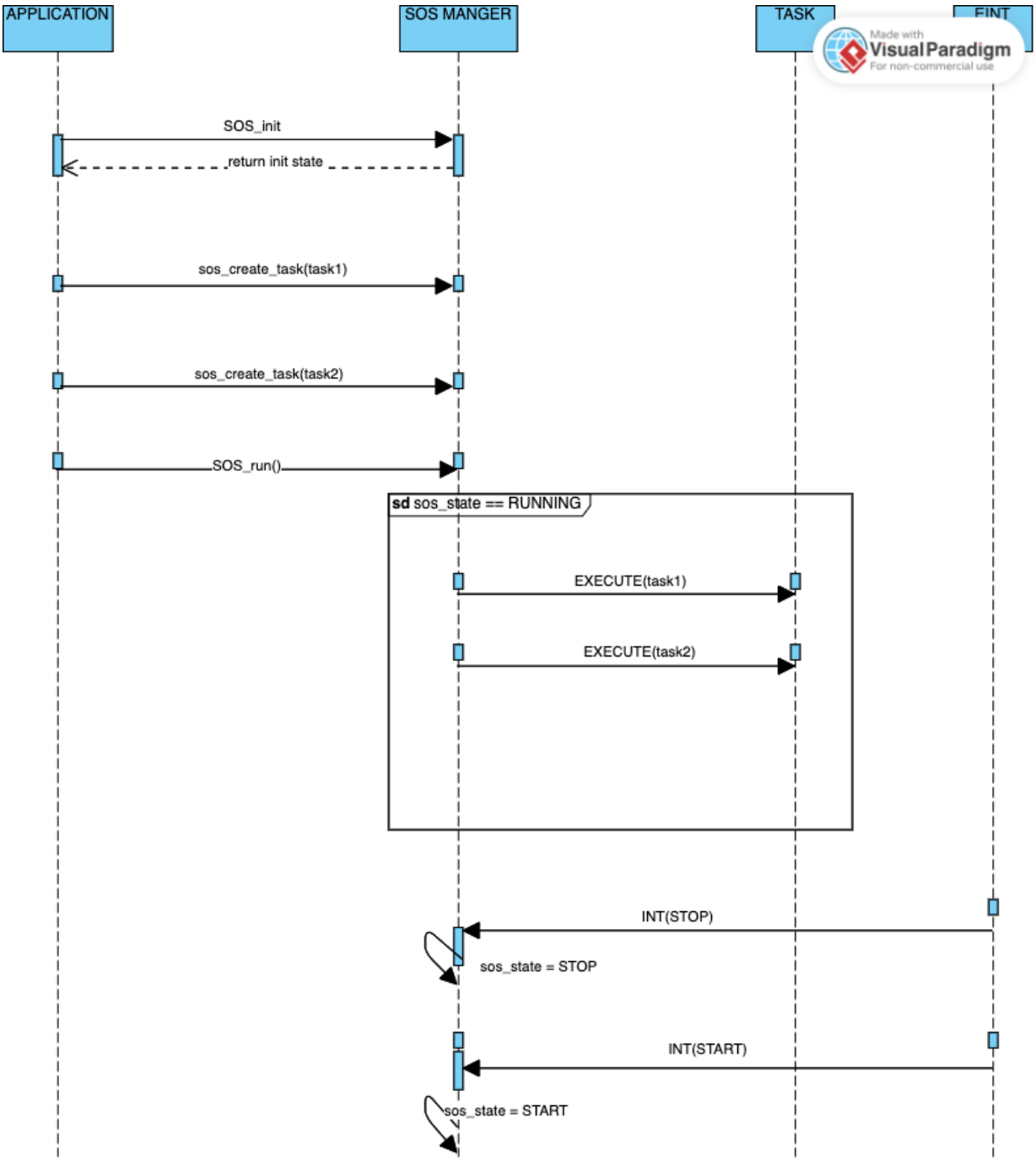
Description: The system is stopped and not executing tasks.

Events:

- PBUTTON0_PRESSED: No action.

- PBUTTON1_PRESSED: Transition to the RUNNING state.

Sequence Diagram


APPLICATION
SOS MANGER
TASK
EINT
Made with Visual Paradigm For non-commercial use
SOS_init
return init state
sos_create_task(task1)
sos_create_task(task2)
SOS_run()
sd sos_state == RUNNING
EXECUTE(task1)
EXECUTE(task2)
INT(STOP)
sos_state = STOP
INT(START)
sos_state = START

Low Level Design:

Flowchart

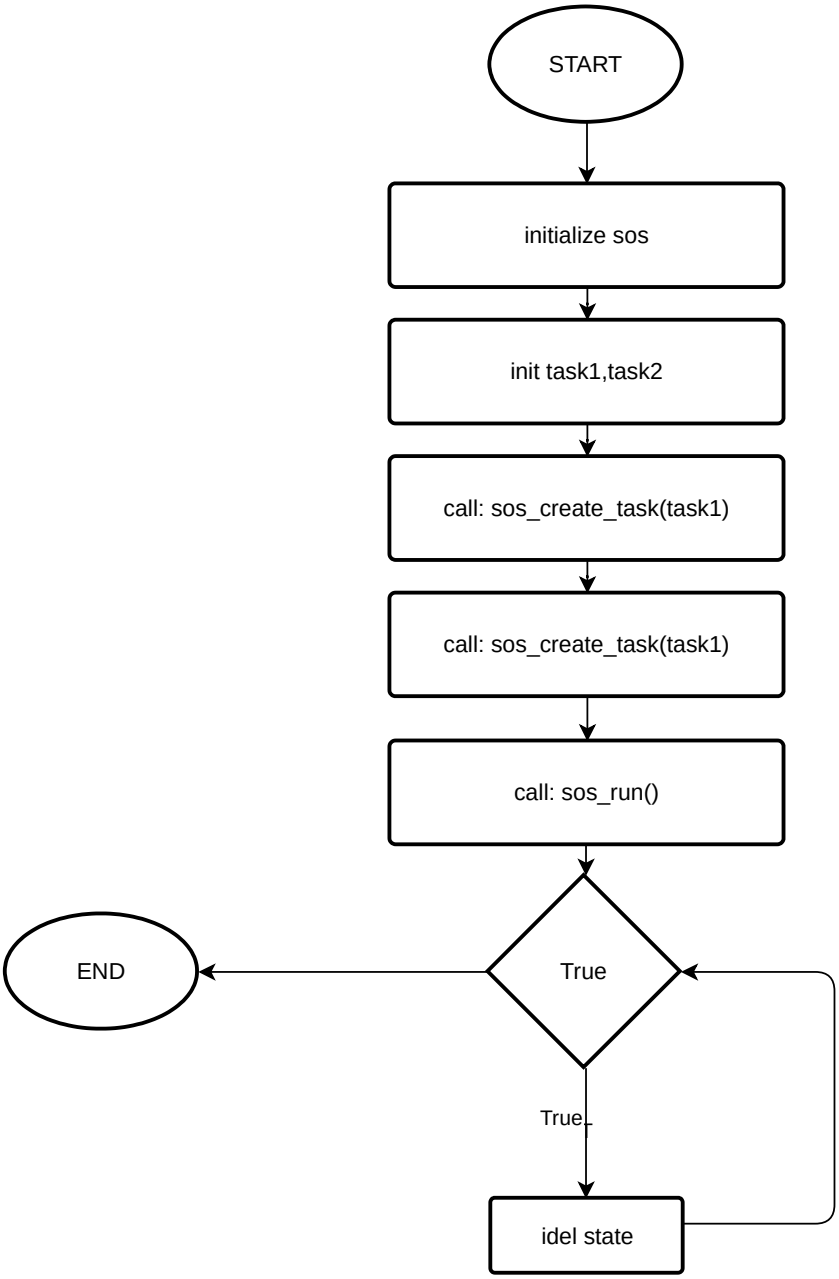
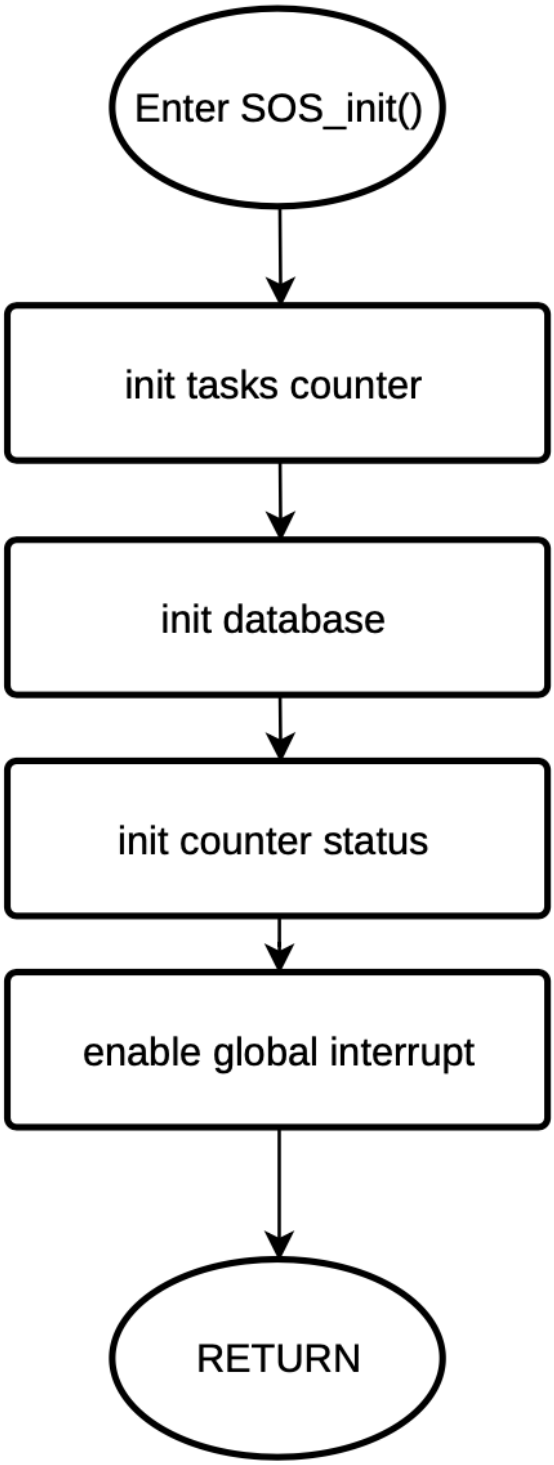
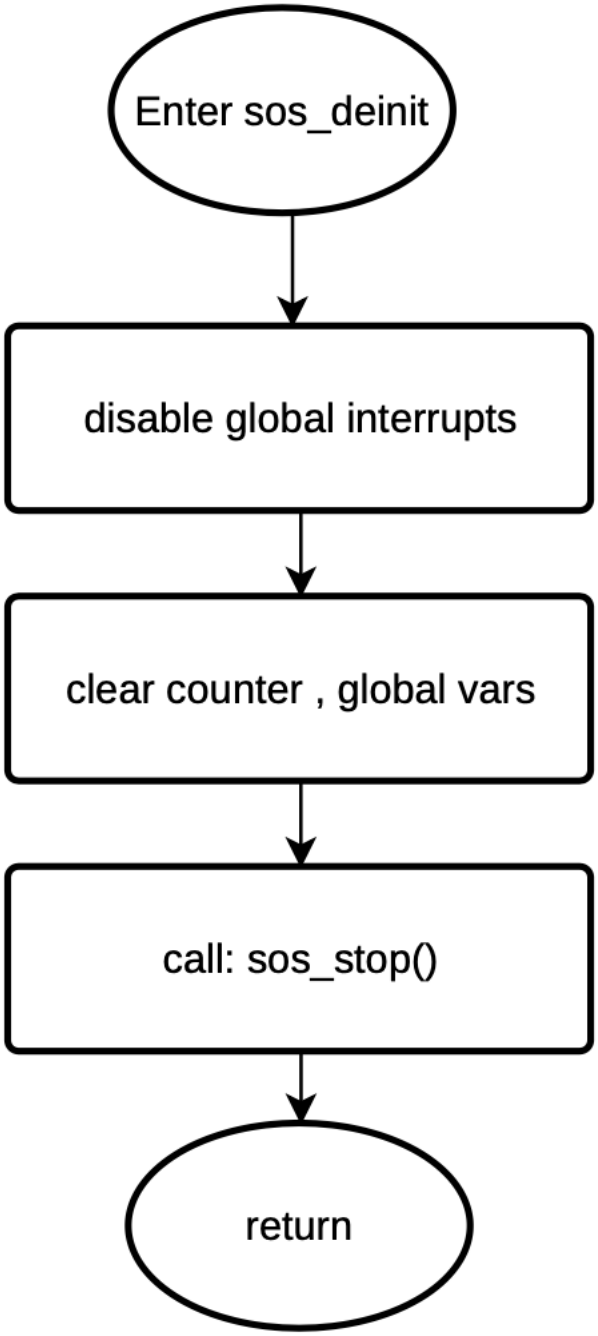
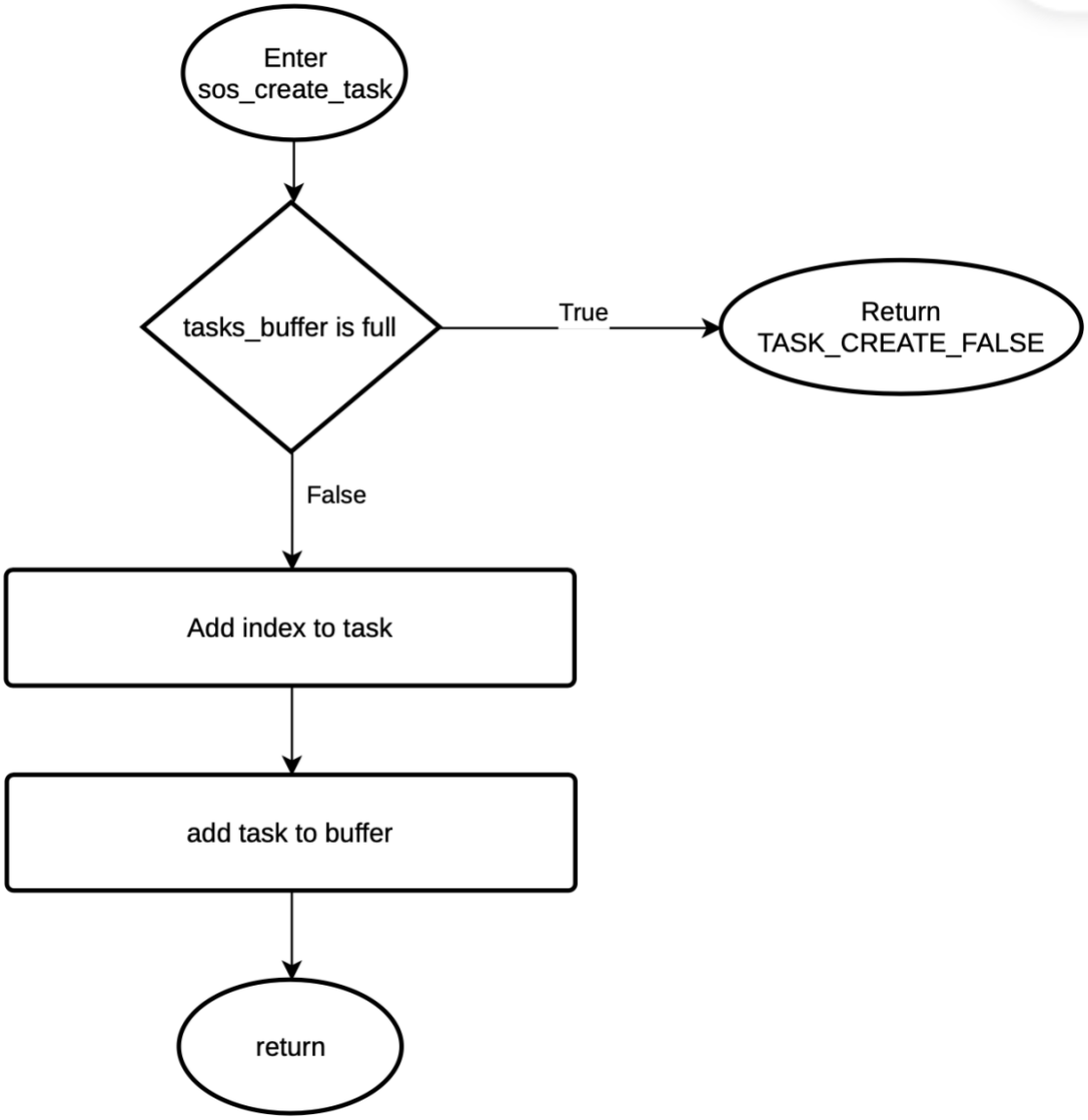
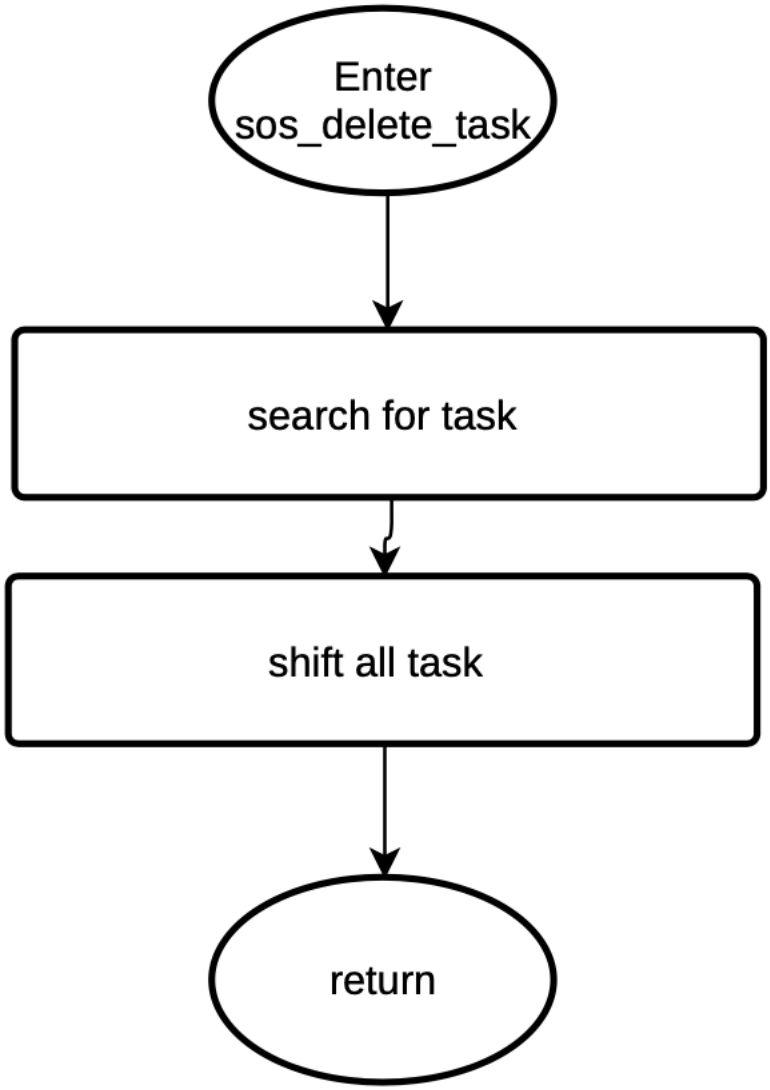


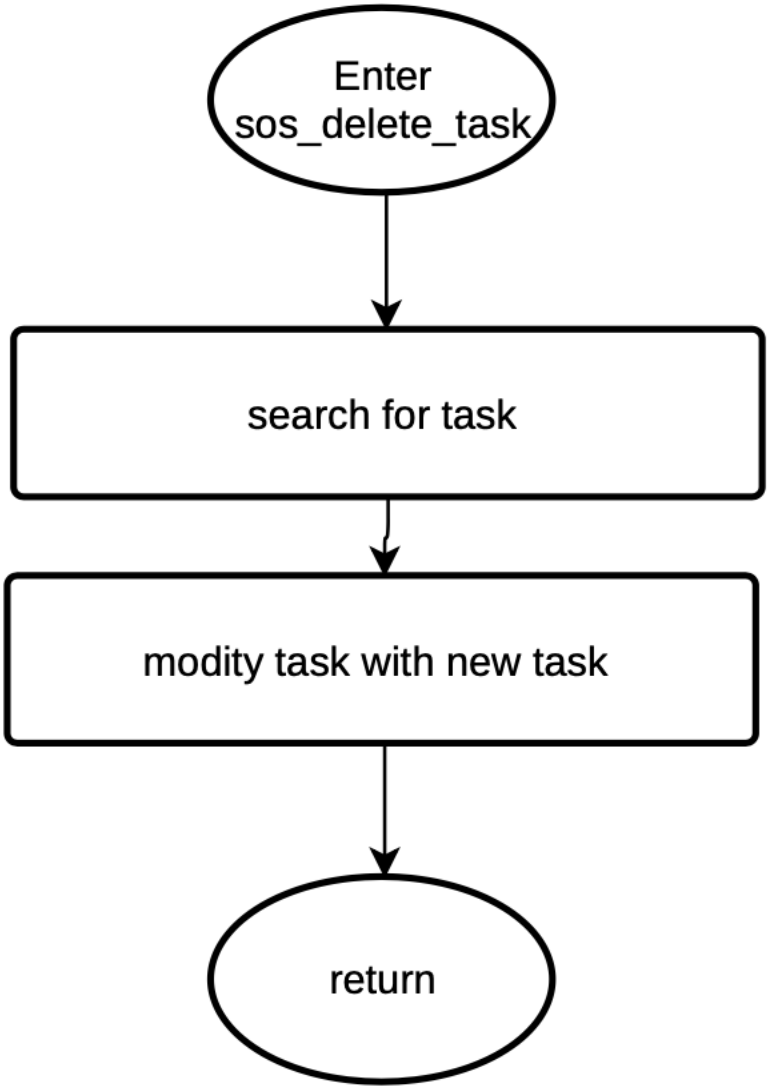
Figure 1 Main Flow Chart

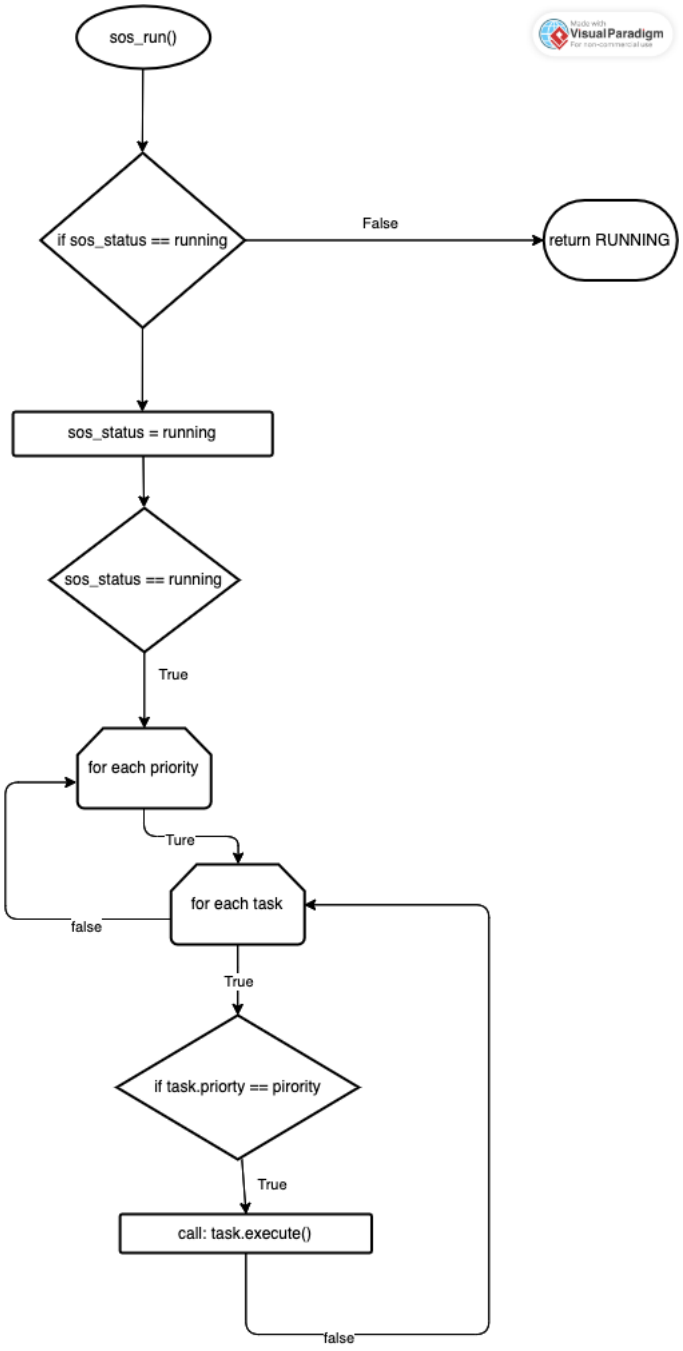

Enter SOS_init()
init tasks counter
init database
init counter status
enable global interrupt
RETURN


Enter sos_deinit
disable global interrupts
clear counter , global vars
call: sos_stop()
return


Enter
sos_create_task
tasks_buffer is full
True
Return
TASK_CREATE_FALSE
False
Add index to task
add task to buffer
return


Enter
sos_delete_task
search for task
shift all task
return


Enter
sos_delete_task
search for task
modity task with new task
return


sos_run()
if sos_status == running
False
return RUNNING
sos_status = running
sos_status == running
True
for each priority
Ture
for each task
false
True
if task.priorty == pirority
True
call: task.execute()
false