Earliest Deadline First

Implementation in FreeRTOS

Authored by: Muhammad Asim Ali

Date: April 24, 2021

Introduction

The earliest deadline first (EDF) scheduling algorithm is a priority-based algorithm that allows the task with the earliest deadline to be scheduled first. The scheduling points for this algorithm are at the arrival of tasks. The EDF algorithm is optimal for a uni-processor machine with pre-emption. FreeRTOS does not have the EDF scheduling algorithm built in natively, the policy supported by FreeRTOS is fixed-priority scheduling assigned by the user.

This project implements the EDF scheduling algorithm in the user space of FreeRTOS. A task titled “DDS” acts as a scheduler to prioritize the task with the earliest deadline first using the vTaskPrioritySet API function to update priorities dynamically. Software timers are used with another set of tasks called the “Task Generators” (one task generator for each user task that is to be scheduled) to signal the arrival of a periodic task. The software timers have the “auto-reload” setting set to true to accommodate the periodic nature of the tasks.

System overview diagram

The following diagram provides an overview of the relationships between the F-Tasks, queues, and DDS functions used in the final version of this project.

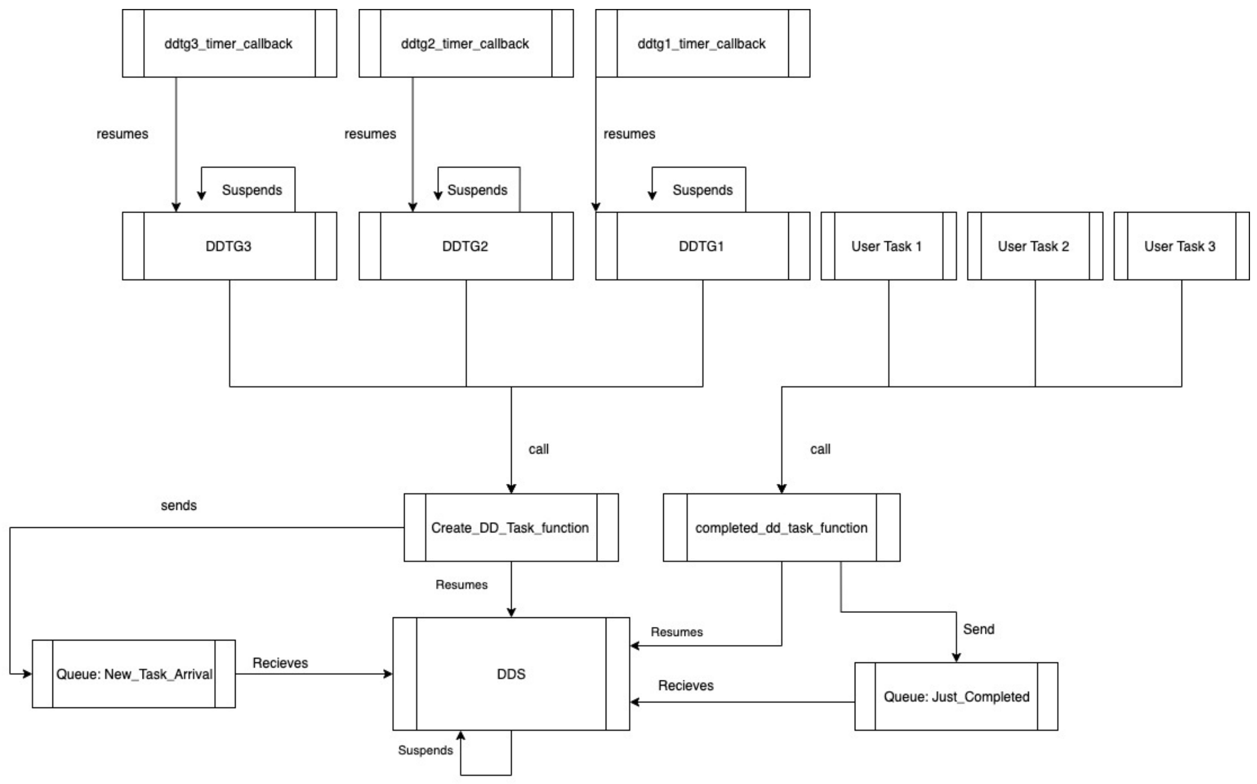

Figure 1 - System overview diagram showing the relationship between the tasks and queues.

Design Document

The pseudo-code below is the Design document that was the initial point of reference for the development of the EDF scheduler. The most significant difference between the initial design and the final system was the task generator. Initially, the design considered implementing a single task generator task for all the F-tasks. However, as development progressed a task generator for each user F-task was created instead due to its ease of implementation.

```
Task: DDS

This task has the highest priority. Is suspended unless a call is made to its interface functions
(the tasks that write messages which are read by this task)

Variables:
A local variable for the active task list
A local variable for the completed task list

Queues:
        New_task_queue;
        Just_completed_task_queue;

Void release_dd_task() {
        Send to New_task_queue the struct containing the new tasks information.

}

Void complete_dd_task()
{
        Send a message to the just_completed_task_queue with the task id of the completed
task
}

Void DDS (){
Wait to receive a message:
        recieveNew_Task_Queue
    -   Assign a release time
    -   Add the task to the active task list
    -   Sort the active task list
    -   Set the priorities of the user-defined tasks with the active task list

Wait to receive a message from just_completed_queue
    -   Receive just_completed_task_queue
    -   Assign a completion time to the newly completed dd task
    -   Remove the dd task from active task list
    -   Add the dd_task to the completed task list
```

- sort the active task list again
- Update priorities of user-defined tasks

}

Task: Deadline-Drive Task Generator

Void deadline_driven_task_generator(){

    - Lower priority than DDS
    - Normally suspended
    - Resumes execution from a software timer callback which are configured to expire based on each DD-Task's period.
    - Calls release dd_task providing it with:
        - TaskHandle_t t_hanlde;
        - Task_type type; (periodic)
        - Task_id;
        - Absolute_deadline // equivalent to the period ?

}

Software Design

This project used:

- 2 queues
    - Just_Completed_message
    - new_task_message

- 7 tasks
    - DDS :Priority 3
    - Deadline-Task generator 1 : Priority = 2
    - Deadline-Task generator 2: Priority = 2
    - Deadline-Task generator 3: Priority = 2
    - User Task 1: Priority (Idle initially and would be updated by the DDS to 1 when user Task 1 needed to run)
    - User Task 2 (Idle initially and would be updated by the DDS to 1 when user Task 2 needed to run)
    - User Task 3 (Idle initially and would be updated by the DDS to 1 when user Task 3 needed to run)
    - The timer svc task from FreeRTOS was assigned the highest priority of 4.
- 3 Timers
    - Task 1 Timer
    - Task 2 Timer
    - Task 3 Timer

Implementation details

The discussion below summarizes the functionality of each of the tasks and queues.
The main

DDS

The DDS task is the scheduler task, it has the second-highest priority of all tasks. The “tmr svc” task was given a higher priority than the DDS to reduce any latency between the timer callbacks which resume the task generators (see the implementation details on task generators for further information).

The structure of the DDS task is:

- Self-suspension
- Receive from the new task queue
- Receive from the completed task queue

The DDS self suspends and is resumed by either the new_dd_task interface function or by the completed_dd_task interface function when a new user task is generated or when a user task has been completed respectively. If the DDS receives a new task it adds it into a list titled “active_list” and sorts that list by ascending order of absolute deadline (ie the first element of the list has the earliest deadline). After sorting the active list, the DDS assigns an “IDLE” priority to all but the very first task in the “active_list”.

If a task has been completed, the task id will be received in the completed task queue. The DDS removes the completed task from the active list, sorts, updates the priorities of the active list, and adds the completed task into either the overdue or the completed list.

DDS interface functions : new_dd_task , completed_dd_task

The completed_dd_task and new_dd_task function are similar in structure:

- They are called from other tasks with some information
- They send the received information to the DDS through queues
- They resume the suspended DDS task

new_dd_task

The task generator functions call the “new_dd_task” with the task handle, absolute deadline, task id, and task type when a new task is ready. The “new_dd_task” packages this information and sends it via the “new_task_message” queue and then resumes the DDS by calling vTaskResume(dds_handle).

completed_dd_task

The user tasks call the “completed_dd_task” function with the task id when a new task is ready. The “new_dd_task” sends this information via the “just_completed_message” queue and then resumes the DDS by calling vTaskResume(dds_handle).

Task Generators: DDTG1, DDTG2, DDTG3

There are 3 task generators, one for each user task.

The task generators each have a priority of 2. They have the second-highest priority of the tasks in the system (third if considering the “timer svc” as a task as well”). The task generators are normally suspended until a timer callback function resumes its execution. The task generators call the “new_dd_task” interface functions with the task handle, absolute deadline, task id, and task type and then self suspend.

User tasks

The user tasks complete their execution for their allotted time and then they call “completed_dd_task” passes it their respective task_id. The user tasks are initialized with “idle” priority. If a user task is at the front of the “active_list” it is assigned a priority of 1 and all other user tasks have their priority made to idle.

Workflow of the entire system:

a. Task scheduler starts
b. It goes to DDS which is the highest priority task
c. DDS suspends itself
d. One of the DDT runs
e. The DDT calls the interface function with info about its respective task
f. The interface function sends a message to the new task queue and resumes the DDS
g. DDS receives the new task and assigns it a release time, adds it to the active list, sorts the active list, and assigns idle priority to all but the task at the 0th element of the active list.
h. DDS checks if there is a message in the Just_completed queue.
    i. If there is a message the DDS assigns the completed task a completion time, and puts it into the appropriate list (overdue or completed list), and updates the active list.
i. Control returns to DDT
    i. DDT self suspends
j. The highest priority task will run until:
    i. A software timer goes off which resumes DDT
        1. The DDT will call the interface function
        2. The interface function sends to q and resumes DDS
        3. DDS will add release time and update the active list
    ii. A task has completed its execution
        1. Measure the time elapsed in a task by:
            a. (will have to add parameters so that the task knows its own task id)
            b. xTaskGetTickCount()
            c. After the execution time is done we call the completion interface function
            d. The completion interface function sends the task id for the task that has just completed to the DDS to receive
            e. The completion interface function resumes the DDS function

Algorithms

Algorithm: DDS Scheduler Flow Chart

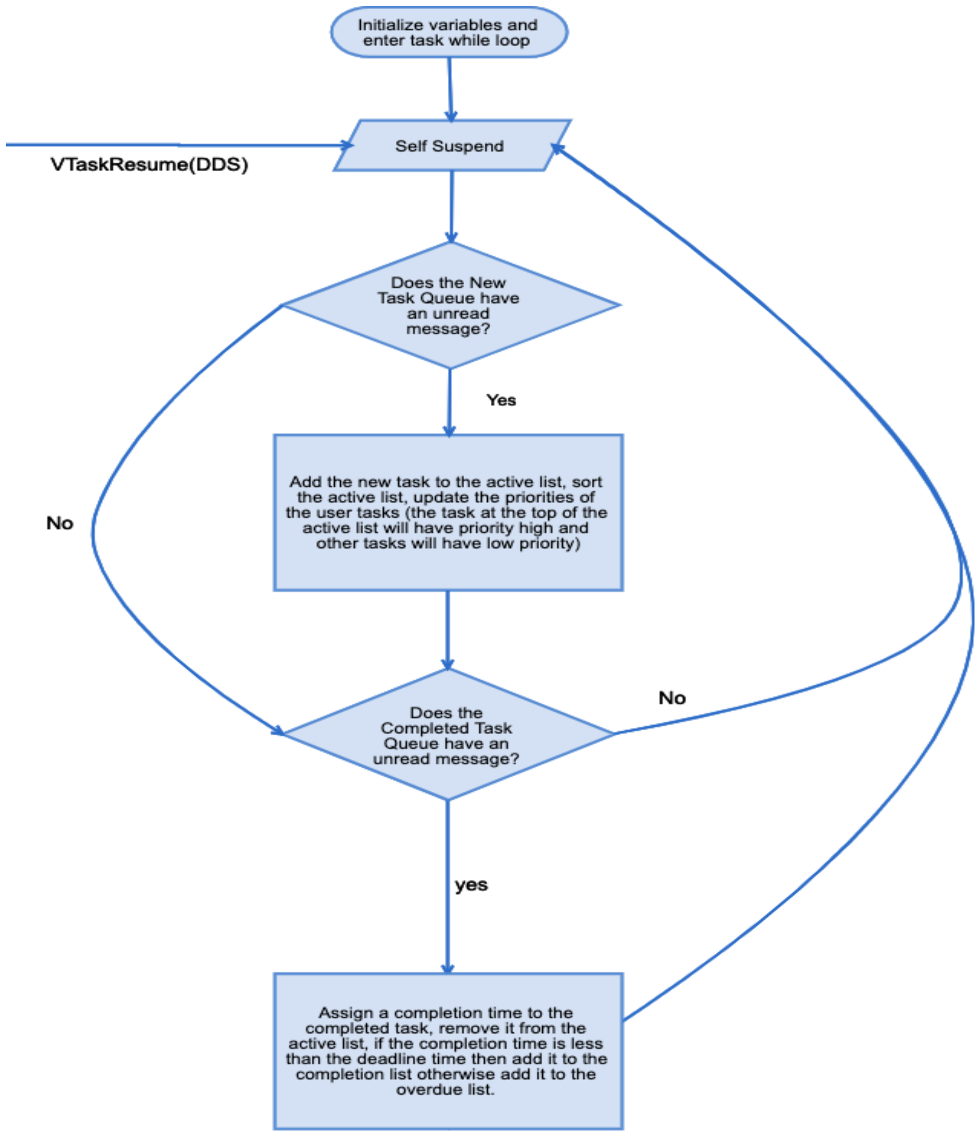
Initialize variables and enter task while loop
VTaskResume(DDS)
Self Suspend
Does the New Task Queue have an unread message?
Yes
No
Add the new task to the active list, sort the active list, update the priorities of the user tasks (the task at the top of the active list will have priority high and other tasks will have low priority)
Does the Completed Task Queue have an unread message?
No
yes
Assign a completion time to the completed task, remove it from the active list, if the completion time is less than the deadline time then add it to the completion list otherwise add it to the overdue list.

Figure 2 - A flow chart showing the algorithm for the DDS.

Algorithm: Deadline-Task Generator

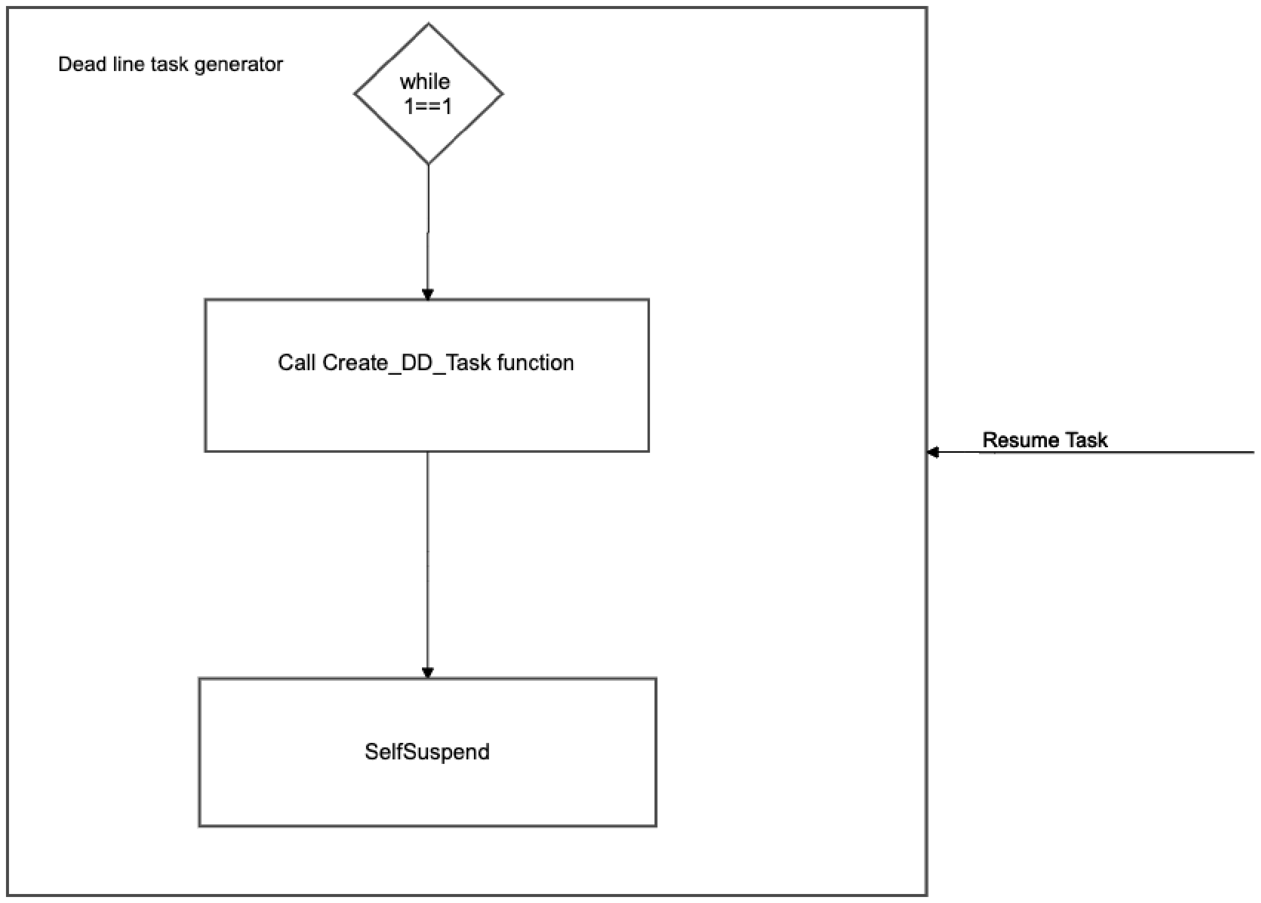

Figure 3 - Deadline Task Generator Flow chart

Limitations and Possible Improvements

The limitations of the current implementation are as follows:

There are problems with the current implementation. In between the DDS, DDT, and interface function executing the system has idle time during which the user tasks also run (since they also have idle priority). An improvement on the current system would be to delete the user tasks and re-create them when they need to be executed again.

Summary

The requirements for the Earliest Deadline First scheduler were implemented using 2 queues and 4 tasks. The queues were used for communicating with the scheduler the scheduling points, task completion, and task release. Software timers were used to call the task generator functions periodically. A task generator was created for each user task.

[1] ECE 455 Lab Manual
[2]https://electronics.stackexchange.com/questions/28091/push-pull-open-drain-pull-up-pull-dow
n