Aperiodic Task Scheduling Using Polling Server

Nitheesh Prakash
MEng Computer Engineering
Virginia Tech
Blacksburg, VA
nitheeshp@vt.edu

Sandeep Errabelly
MEng Computer Engineering
Virginia Tech
Blacksburg, VA
sandeep96@vt.edu

***Abstract*—Our project Aperiodic task scheduling using polling server is to implement a polling server as a periodic task that executes periodically and schedules the aperiodic tasks which are injected at random times. The server will check the aperiodic task queue and schedule the available aperiodic tasks. If no aperiodic tasks are available, next the server will exit its function block. We have evaluated how periodic task load affects response time of aperiodic tasks and also evaluated how the priority(time period) of polling server affects the response time of aperiodic tasks.**

Keywords: Aperiodic, Polling server, FreeRTOS, Response Time, RM Scheduling

I. Introduction

Real-time systems are systems where the functionality of the system not just depends on the logical results, but also depends on the time constrain of each tasks in an application. Real-time scheduling algorithms must support hybrid task sets with both periodic tasks and aperiodic tasks. In periodic tasks, at regular intervals jobs are released. It is characterized by an arrival time, time period, execution time, deadline, and worst-case execution time.

Aperiodic tasks are invoked by the occurrence of an event and doesn't have a fixed arrival time. The task scheduling algorithms must make sure that the deadlines of periodic tasks are met and also the aperiodic tasks are served with less response time.A periodic polling server to check if there is any aperiodic tasks which are released and executing them is a simple yet efficient way of scheduling the hybrid task sets.

II. Background Study

A. Periodic Task

In Real-Time systems, most tasks are the same jobs which are performed periodically. For example, a task of collecting data from a energy meter will be done with an interval of n time units. This type of tasks which is performed repeatedly is called as periodic tasks.

Often periodic tasks are denoted with 4 major entities namely phase, time period, worst-case execution time (WCET) and deadline.

The phase can also be known as the release time of the task, which denotes the arrival time of that particular task or job. Similar time period denotes the periodic time in which the task arrive periodically. The worst-case execution time denotes how long the task will execute. The deadline as the name denotes specifies the time before which the task should complete.

B. Aperiodic Task

In Real-Time systems, some tasks can performed only once and not periodically. For example, a task of which performs data transfer when typing keyboard. This type of tasks which are performed once they arrive or created is called as aperiodic tasks. These tasks are scheduled in the CPU slack time or when there is no periodic task available for scheduling. These tasks do not have a phase, period or deadline. As these tasks arrive when they are created, there is no phase.

C. Polling Server

Polling Server is a periodic tasks which executes for periodically in a specific interval to execute any aperiodic tasks. Polling Server has a queue implemented in it which will contain the aperiodic tasks if there is any. When the polling server periodic task starts to executes, it iterates over the queue to execute the aperiodic tasks available in the queue.

D. Rate-Monotonic Scheduling

In the Rate-Monotonic (RM) scheduling algorithm priorities are assigned according to the cycle duration of the tasks. Tasks with the shortest cycle duration will be assigned the highest priority. [1]

III. Implementation

We have implemented the aperiodic task scheduler by periodically checking the list of aperiodic tasks being injected and scheduling them based on the rate-monotinic scheduling algorithm. For periodically checking the aperiodic tasks list we have used a polling server task which by itself is a periodic task. All the tasks, both periodic and aperiodic tasks, are created at time zero when the application starts. As we have used RM scheduling algorithm all the periodic tasks are executed as per their corresponding time periods. So by varying the time period of our polling server we have varied the priority of scheduling aperiodic tasks. [2]

A. Aperiodic Tasks

Similar to periodic tasks, we have used a control block structure, *xAperiodicTaskControlBlock*, for managing aperiodic tasks. We have created an array, *xATCBQueue[]*, to maintain the aperiodic tasks queue. As new aperiodic tasks are injected, they are added to the queue and removed when a task is executed. We have used head and tail indexes to keep track of where to add a new task and where to fetch a task to execute. We also used to counter, *uxAperiodicTaskCounter*, as an easy way to find how many aperiodic tasks are yet to be scheduled.

prvGetNextAperiodicTask() will check the counter if there are any aperiodic tasks and then returns the task which is pointed by the head index. *prvFindEmptyElementIndexATCB()* will return the tail index and increment the tail index. *vSchedulerAperiodicTaskCreate()* will get an empty slot from the aperiodic task queue by calling prvFindEmptyElementIndexATCB() and fills all the parameters of the aperiodic task. Then the uxAperiodicTaskCounter is incremented.

B. Polling Server

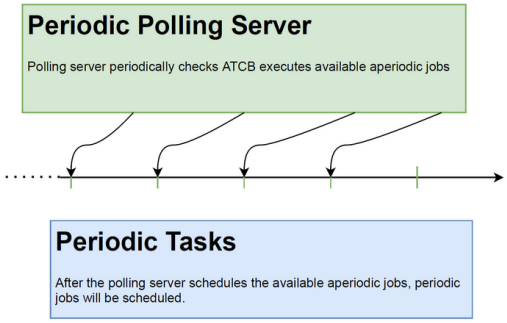

Fig. 1. Polling Server

prvPollingServerCreate() will create the polling server as a periodic task. It calls prvFindEmptyElementIndexTCB() to get an empty slot from the periodic tasks array and fills all the parameters about the polling server. Polling server will execute just as another periodic task(Fig. 1) and executes the aperiodic tasks if available.

prvPollingServerFunction() will get the next aperiodic task from the queue by calling prvGetNextAperiodicTask() and runs its task code. The task counter is also decremented and response time is calculated once the task code execution is complete.

When the polling server misses its deadline and gets executed on next time period, it will just continue to execute the aperiodic task which was being executed when the deadline miss occurred. We are not recreating the polling server when its deadline is missed as opposed to other periodic tasks.

C. Response Time Calculation

We have recorded the Response time of the aperiodic tasks in terms of tick counts. It is calculated by saving the tick count when the task has arrived and subtracting it with the tick count of when its task code is executed fully. Start time is recorded from vSchedulerAperiodicTaskCreate() when every task is created and stored in a TCB structure parameter associated with that task and end time is recorded from prvPollingServerFunction() where the task code is executed.

$$ResponseTime = CompletionTime - ArrivalTime$$

IV. Evaluation and Results

We have evaluated our implementation of polling server with various experimental settings such as evaluating for different periodic loads, for different polling server setting and by varying the worst-case execution time of polling server. All these different testing setting and their respective experimental results have been explained in the below sections.

A. Experimental Setup

We have used Arduino Mega 2560 as our test-bench hardware and to program the board we used Arduino IDE 1.8.19. In order to test our implementation we have used FreeRTOS 10.4.3-6 [3]. We have used 3 periodic tasks and 4 aperiodic tasks as our taskset for testing our polling server implementation. We have used aperiodic jobs as our polling server supports aperiodic job executions.

TABLE I
Task Set

Task Name	Time Period	WCET
T_1	200	94
T_2	500	99
T_3	600	199

Table I shows the task set we have used to test our implementation. Table I also shows the task set with respective WCET. The Utilization of the above taskset in the table I is 1. The WCET of the aperiodic tasks are set to 50ms for all the 4 aperiodic tasks which are being used in our evaluation. Even though all the aperiodic tasks arrive at same time, the tasks will be executed based on its place in the polling server queue. We are using Rate-Monotonic(RM) scheduling algorithm to schedule the periodic tasks.

B. Aperiodic Response Time vs Periodic Load

We are running three periodic tasks and injecting four aperiodic tasks whose details are mentioned in the above section. In order to evaluate the performance of our polling server, we have decided to test our implementation against various periodic load ranging from 0 to 1 with the step increase of 0.1. For this evaluation, we have set out polling server period as 600ms and the worst-case execution time as 200ms. The main objective of our evaluation here is to find the impact

of periodic load on the response time of the aperiodic tasks. In this case, since the polling server period is 600ms, the priority of the polling server will be the lowest priority task.

In order to evaluated for various periodic load with a step of 0.1, we have decided to vary the WCET of each task. For example, For task T1, the WCET for 1 utilization is 94, which is for 1 periodic load the WCET is 94. In order to get the WCET for 0.9 periodic load, multiply 0.9 to the original WCET 94, 85 is the new WCET for 0.9 periodic load. Similarly, the WCET of each periodic with respective to each periodic load is calculated. The calculation model and framework is provided with the attached xls file.

The below Table II shows the response time of each aperiodic task for various periodic loads. The Fig 2 shows the graphical representation of the Table II. From the graph we can see that the response of each aperiodic task increases with the increase in the periodic load and vise versa. This is because the priority of the polling server being the lowest, with the increase periodic load, we can say that there are more higher priority tasks waiting to be scheduled. So the response time of aperiodic tasks increases.

TABLE II
RESPONSE TIME OF EACH APERIODIC TASK FOR DIFFERENT PERIODIC LOADS

Periodic load	Aperiodic Task's Response Time in Time Ticks			
	Task 1	Task 2	Task 3	Task 4
0	1	3	4	6
0.1	3	5	6	8
0.2	4	6	8	9
0.3	6	7	9	11
0.4	7	9	10	14
0.5	8	10	11	14
0.6	9	11	15	16
0.7	11	17	18	20
0.8	14	16	17	19
0.9	16	18	20	21

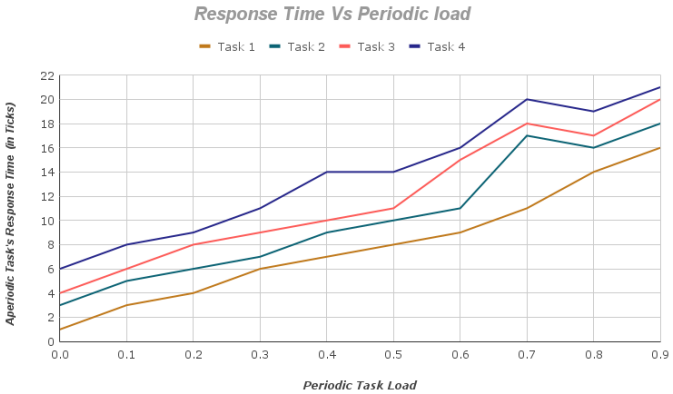

Fig. 2. Response Time Vs Periodic Load (PS period 600 ms)

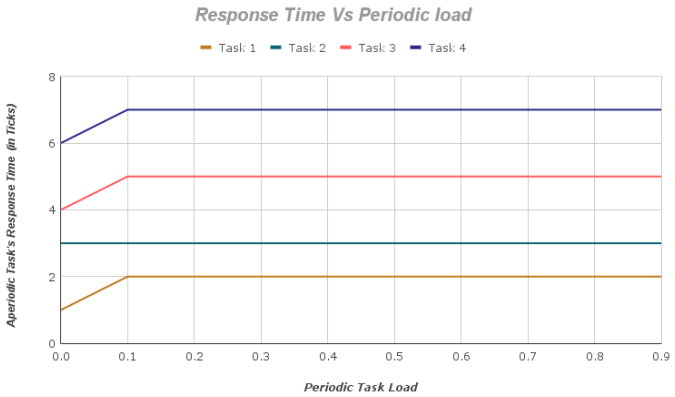

Fig. 3. Response Time Vs Periodic Load (PS period 100 ms)

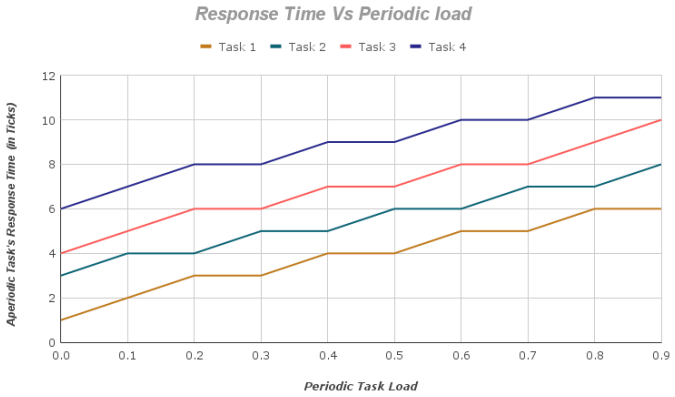

Fig. 4. Response Time Vs Periodic Load (PS period 400 ms)

C. Aperiodic Response Time vs Polling Server Period(Priority)

To evaluate how the priority of the polling server affects the response time of each aperiodic task we have done this experiment. As we are using RM scheduling where the time period of each task will be used to determine the priority of each task. We used 100ms, 400ms, 600ms and 1200ms as our test time periods for polling server as shown in Table III. Also we varied the load from 0 to 1 in steps of 0.1 and took the average of all the response time for each aperiodic task to obtain the average response time and visually represented it. As the time period of polling server is increased its priority will decrease and from the table and graph we can see that as priority decreases the response time of each aperiodic tasks are increasing as expected.

We have also evaluated the average response time of each aperiodic tasks for various periodic loads with respect to each polling server period and is graphically represented in the Fig 6. We can visually see impact of the polling server priority or period affecting the average response time of the aperiodic tasks.

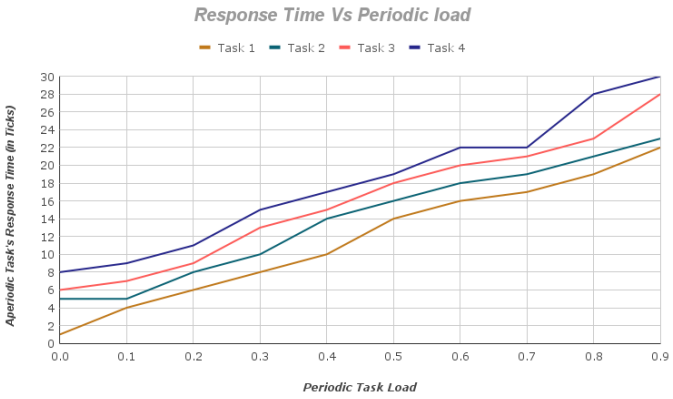

Fig. 5. Response Time Vs Periodic Load (PS period 1200 ms)

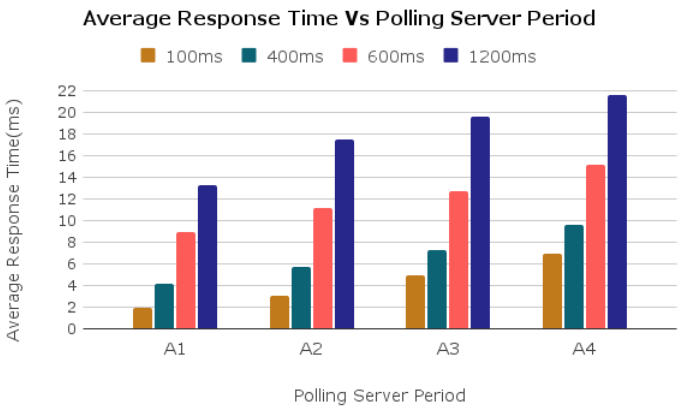

Fig. 6. Average Response Time each aperiodic task with varying polling server priority

When the polling server's period is 100ms (i.e., Highest priority) is for the polling server, in the Fig 3, we can see that the response of each aperiodic tasks is similar to 0 periodic load because of the polling server task being highest priority, this task executes first.

When the polling server's period is 1200ms (i.e., Lowest priority) is for the polling server, in the Fig 5, we can see that the response of each aperiodic tasks is varies lot because of the polling server task being lowest priority and the next occurrence of the polling sever is not until 1200ms, so the response time of aperiodic task which did not executes its previous occurrence has to wait for another 1200ms.

Ideally for a 100% (1) periodic load and the polling server being the lowest priority, aperiodic should never happen, but because our systems are not ideal, we can see the response time of aperiodic tasks.

Fig. 7 show the total response time of aperiodic tasks under different periodic load and different polling server periods. This show the summary of our complete evaluation.

TABLE III
COMPARISON OF TOTAL RESPONSE TIME OF THE APERIODIC TASKS FOR DIFFERENT PERIODIC LOAD AND FOR DIFFERENT POLLING SERVER PERIOD

Aperiodic Tasks	Periodic Server Period in ms			
	100	400	600	1200
Task 1	1.91	4.18	8.91	13.27
Task 2	3	5.73	11.18	17.45
Task 3	4.91	7.27	12.73	19.54
Task 4	6.91	9.64	15.18	21.54

D. Aperiodic Response Time vs Polling Server WCET

We have also tested our implementation with the same experimental setup mentioned in the section In order to check the impact of the polling server's WCET on the aperiodic average response time, we decided to keep the polling server's

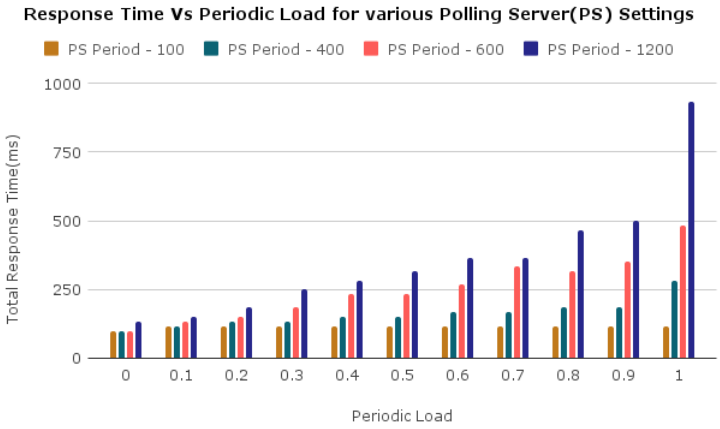

Fig. 7. Comparison of Total Response Time of the Aperiodic tasks for different periodic load and for different polling server period

period to constant as 600ms and vary the WCET(worst-case execution time) as 100, 200, 400. We could see through this experiment that more of the WCET available for the polling server, the number of aperiodic tasks executed before preemption is more than the number of aperiodic tasks executed for lesser WCET values. The preemption solely depends on the priority of the polling server which is in-turn depended on the period of polling server as the scheduling algorithm used here is Rate-Monotonic (RM) scheduling algorithm.

V. CONCLUSION

We have implemented aperiodic task scheduler using polling server based on RM scheduling and evaluated how response time of the aperiodic tasks vary with different periodic task loads and also with different polling server priorities. As expected when the periodic task load increases the response time of the aperiodic tasks increases as the polling server gets less time to execute. And as polling server priority is decreased, other periodic tasks are first scheduled and hence

the response time of the aperiodic tasks goes up as expected. Hence we can conclude that our implementation works as expected.

REFERENCES

[1] Kase, R. (2016). (thesis). Efficient Scheduling Library for FreeRTOS. Retrieved May 7, 2022, from http://www.diva-portal.org/smash/get/diva2:1085303/FULLTEXT01.pdf.

[2] Kase, R. (2016). ESFree. GitHub. Retrieved May 7, 2022, from https://github.com/RobinK2/ESFree

[3] RTOS - free professionally developed and robust real time operating system for small embedded systems development. FreeRTOS. (2022, January 26). Retrieved May 7, 2022, from https://www.freertos.org/RTOS.html