Cyclic Executive Scheduling (Time Triggered Approach)

Embedded Systems Project

Participant Name: Siraj Tayyab Khan & Zain-ul-Abidin

Participant Roll: MSEE-22002 & MSEE-22007

University: Information Technology University, Lahore

EE-580: Embedded Systems Design

Instructor: Dr. Rehan Ahmed

Teaching Assistant: Emad Ali & Rabia Ahmed

May 25th, 2023

~ (Content) ~

~ (Cyclic Executive Scheduling (Time Triggered Approach)) ~

Introduction:

In the world of Embedded Systems, we define a mechanism or a set of rules by which our embedded device handles various kinds of tasks, and we call it a scheduling scheme. We opt for a particular scheduling scheme based on our scenario of tasks and requirements that we want to meet. Generally, we can classify embedded scheduling scheme based on static nature, periodicity, pre-emption, and dependence like table of some scheduling algorithms, we discussed thoroughly in this course. But despite mentioning all of the given below algorithms, we are particularly interested in **Time Triggered Cyclic Executive Scheduling (TTCES)** for our project.

Algorithm	**Static/Dynamic**	**Pre-emptive**	**Periodic/ Aperiodic**	**Dependency**
Simple Time **(STT)**	Static	Not	Periodic	No
Time Triggered Cyclic Executive **(TTCE)**	Static	Not	Periodic	No
Earliest Deadline **Due (EDD)**	-	Not	Aperiodic	No
Earliest Deadline First **(EDF)**	Dynamic	Yes	Aperiodic	No
Earliest Deadline First Star **(EDF*)**	Dynamic	Yes	Aperiodic	Yes
Rate Monotonic **(RM)**	Static	Yes	Periodic	No
Deadline Monotonic **(DM)**	Dynamic	Yes	Periodic	No
Rate Monotonic Polling Server **(RMPS)**	Dynamic	Yes	Both	No
Total Bandwidth Server **(TBS)**	Static	Yes	Both	No
YDS Online Dynamic Voltage Frequency Scaling **(DVFS)**	Dynamic	-	-	-
YDS Offline Dynamic Voltage Frequency Scaling **(DVFS)**	Static	-	-	-

Figure 1.1 Table of comparison

Cyclic Executive Scheduling (CES):

Cyclic Executive Scheduling is needed when we want to schedule a taskset (multiple tasks), such that ***each task belonging to the taskset can have different time period cycle of execution to the other member belonging to the same taskset.*** Secondly, taskset needs to respect deadlines for all its members.

Before discussing more about CES, let us **define a task**. A task is a member of taskset with its task number $\boldsymbol{i}$, and instance $\boldsymbol{j}$ where Arrival time is denoted by $\boldsymbol{r_{i,j}}$, Computation time $\boldsymbol{c_i}$, Deadline $\boldsymbol{D_i}$, Phase $\boldsymbol{\varphi_i}$, and Time period $\boldsymbol{T_i}$.

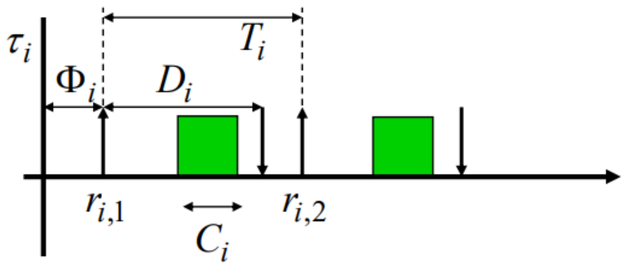


Figure 1.2 Example of a Task Schedule

In CES, we first calculate hyper period $\boldsymbol{P}$ of the taskset by taking Least Common Multiple (LCM) of all periods of the taskset. Special thing about hyper period is that if we are able to schedule all tasks, and their corresponding instances within the hyper period, then we can just repeat that schedule till forever without needing to calculate any further.

Then we divide whole hype period into frames of frame size $\boldsymbol{f}$, where each frame has at least one or more task scheduled in it. There are other constraints for the frame size as well like a frame size cannot be lesser than the compute time of any member of the taskset, and on the flip side it cannot exceed the time period of any member of the taskset.

$$\boldsymbol{C_i \,\forall\, tasks\; \tau_i \leq f \leq\; T_i \,\forall\, tasks\; \tau_i}$$

Lastly for the frame size that it should always be a positive integer multiple of the hyper period.

$$\boldsymbol{P = nf : n \in Z^+}$$

There should be at least one full frame between the deadline and arrival of the task.

$$\boldsymbol{2f - gcd(T_i, f) \leq\; D_i \,\forall\, tasks\; \tau_i}$$

Last part is that once you have somehow generated your schedule it should not violate things like an instance cannot start before its arrival, and deadlines should be respected irrespective of the phase.

Example with 4 tasks:

- $\tau_1 : T_1 = 6, D_1 = 6, C_1 = 2$ $\tau_2 : T_2 = 9, D_2 = 9, C_2 = 2$
  $\tau_3 : T_3 = 12, D_3 = 8, C_3 = 2$ $\tau_4 : T_4 = 18, D_4 = 10, C_1 = 4$
- $P = 36, f = 4$

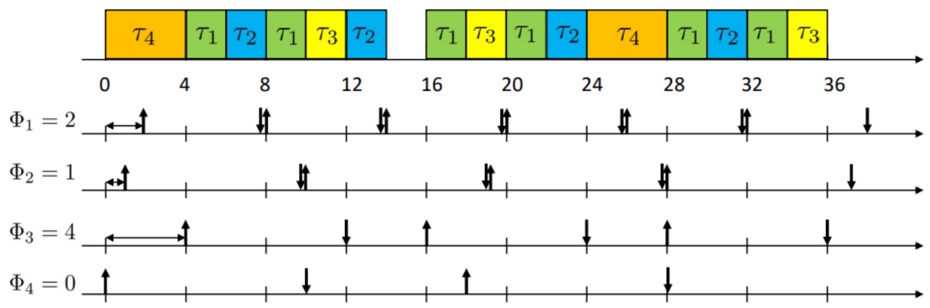


Figure 1.3 Example of a Taskset Schedule

Implementation:

In this project we are meant to design a system such that it takes taskset as an input, and schedules that taskset using CES algorithm. To implement this our **language of choice is Python**, and **format of delivery is Jupyter Notebook file**. Rest is left on the physical implementation, which can be done using FreeRTOS, on some microcontroller supporting it such as STM32F401CCU6 or any ESP32 board. But scope of this project revolves around the periphery of schedule generation rather than the deploying phase on some microcontroller, which is pretty much the same like implementation of any other scheduling algorithm. Although we did not follow any hard software design methodology, but closest representation is **Iterative Waterfall Model**.

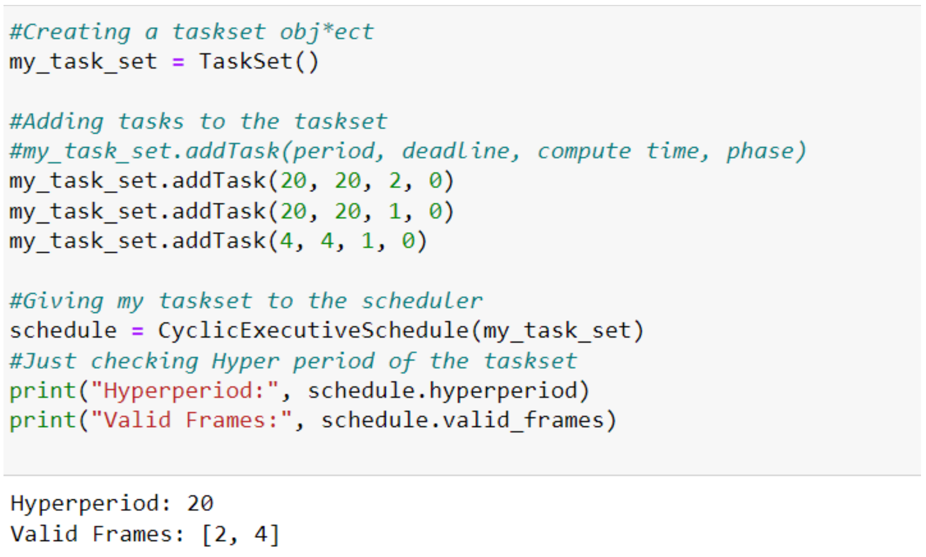
In case of **Algorithm**, initial method is same like It starts by receiving tasksets, then it does initial basic tests like checking overall utilization should not cross hundred percent. After scheduler receives the taskset then it computes the hyper period and valid frames. Valid frames are computed using conditions described in the CES description.

After getting valid frame sizes, real challenge of scheduling begins. Despite getting valid frame sizes does not guarantee the schedule. So, to schedule our taskset, **a bit map or bool array block** is created of **3-dimensions,** where dimensions are **Tasks** by **Instances** by **Frames**. True value represents that jth instance of ith task is scheduled in kth frame. On a supporting note, that this bit mapping is generated by taking care of time periods, deadlines, and phases of the taskset.

In the last part once we have the bit map, we can create our own algorithm that picks bit map frame by frame. For each frame first it deciphers what tasks instances qualify to be in first frame. After getting all qualifying task instances, then it checks what are the tasks instances that can be scheduled in our current frame but not in the next frame are assigned as a constraint to be mandatory tasks for the current frame. After getting mandatory tasks, and other tasks that can be scheduled in the current frame, it tries to generate a combination that does not miss mandatory tasks, and combination should schedule maximum number of tasks to fit in the frame without violation of frame sizes. After schedule a particular task instance in the current frame, it marks the same instance in the other frames as false, which in other words mean that you do not want to schedule same instance of the task again. Then we repeat that all steps for all frames, and in the end, we check our bit map if there is any task instance left unscheduled. In case if whole bit map tells you false then it means that all tasks' instances are scheduled with the given frame size, and schedule is exported in csv file format for the user convince.

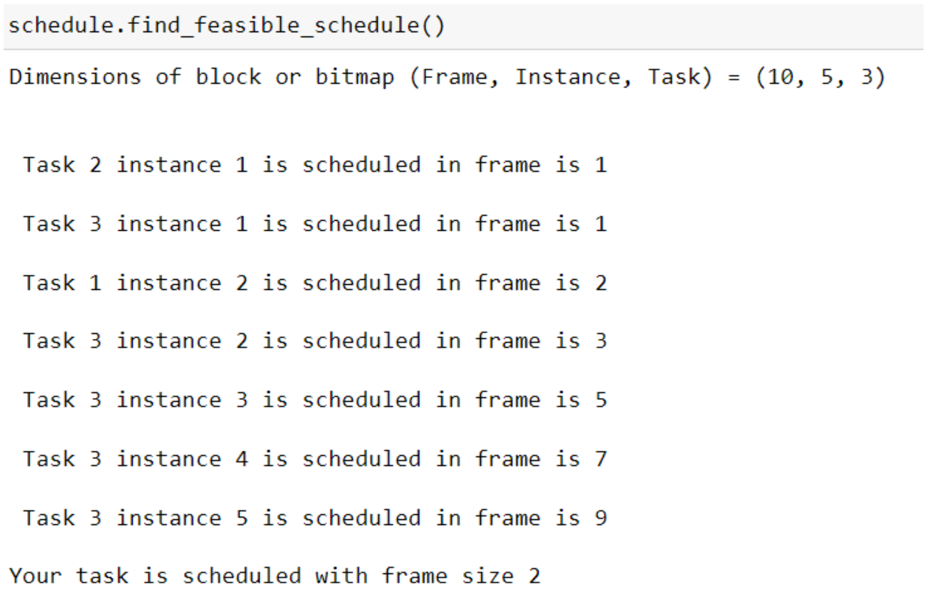
Results:

In demonstration of scheduling scheme, I created taskset object, and later add more tasks in the taskset. Then CES scheduler would receive a taskset object as input to create an object of scheduler. Then scheduler is told to find a feasible schedule which will create .csv format file or files for every frame size it could schedule. Then later we can use function of plot schedule to draw plot for all scheduled frame sizes, and export a .png image file with labelled output schedule as snippets attached below.



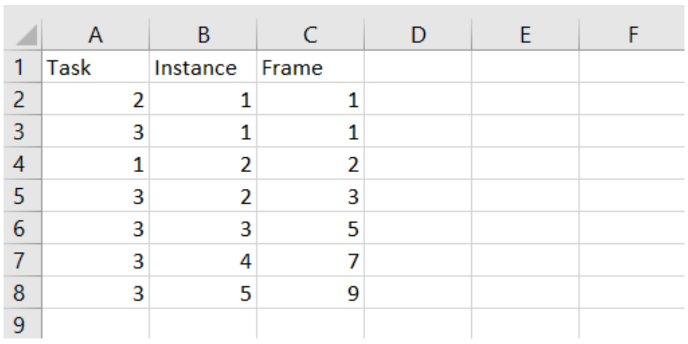
```
#Creating a taskset obj*ect
my_task_set = TaskSet()

#Adding tasks to the taskset
#my_task_set.addTask(period, deadline, compute time, phase)
my_task_set.addTask(20, 20, 2, 0)
my_task_set.addTask(20, 20, 1, 0)
my_task_set.addTask(4, 4, 1, 0)

#Giving my taskset to the scheduler
schedule = CyclicExecutiveSchedule(my_task_set)
#Just checking Hyper period of the taskset
print("Hyperperiod:", schedule.hyperperiod)
print("Valid Frames:", schedule.valid_frames)
```

```
Hyperperiod: 20
Valid Frames: [2, 4]
```



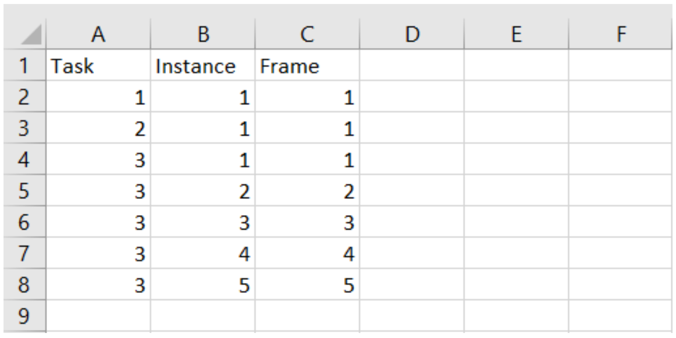
```
schedule.find_feasible_schedule()
```

```
Dimensions of block or bitmap (Frame, Instance, Task) = (10, 5, 3)


 Task 2 instance 1 is scheduled in frame is 1

 Task 3 instance 1 is scheduled in frame is 1

 Task 1 instance 2 is scheduled in frame is 2

 Task 3 instance 2 is scheduled in frame is 3

 Task 3 instance 3 is scheduled in frame is 5

 Task 3 instance 4 is scheduled in frame is 7

 Task 3 instance 5 is scheduled in frame is 9

Your task is scheduled with frame size 2
```



	A	B	C	D	E	F
1	Task	Instance	Frame			
2	2	1	1			
3	3	1	1			
4	1	2	2			
5	3	2	3			
6	3	3	5			
7	3	4	7			
8	3	5	9			
9						

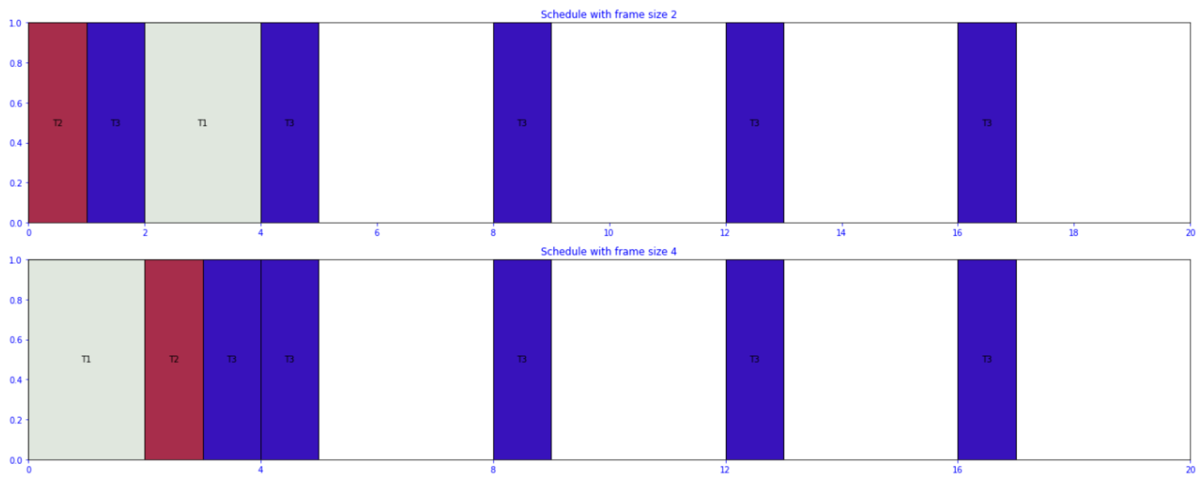
```
Dimensions of block or bitmap (Frame, Instance, Task) = (5, 5, 3)

 Task 1 instance 1 is scheduled in frame is 1

 Task 2 instance 1 is scheduled in frame is 1

 Task 3 instance 1 is scheduled in frame is 1

 Task 3 instance 2 is scheduled in frame is 2

 Task 3 instance 3 is scheduled in frame is 3

 Task 3 instance 4 is scheduled in frame is 4

 Task 3 instance 5 is scheduled in frame is 5
Your task is scheduled with frame size 4
```



	A	B	C	D	E	F
1	Task	Instance	Frame			
2	1	1	1			
3	2	1	1			
4	3	1	1			
5	3	2	2			
6	3	3	3			
7	3	4	4			
8	3	5	5			
9						

Final Schedule Plot:

```
schedule.plotSchedule()
```


Schedule with frame size 2
T2
T3
T1
T3
T3
T3
T3
Schedule with frame size 4
T1
T2
T3
T3
T3
T3
T3