Implementation:

All the hardware components are implemented as a separate classis, for convenience. Scheduler is implemented in a separate file following the separation of concerns principle. Scheduler.c contains only one public function which must initiate the scheduler. The main private function of the class 'vScheduler' is a task that gets started by the initiator function, and never suspends. The interface is written in such a way that the class is easily extendible with different types of scheduling algorithms. vScheduler's purpose is to make sure that a periodic task gets initiated at the right time. Then, based on the scheduling function that implements a certain scheduling algorithm, vScheduler task receives a task that should be run next. Then, vScheduler function starts that task. vBrewer is a function that gets run by the vScheduler. It is being provided with the task handler which allows it to manage the running task state. A semaphore is used to control the execution of the functions and prevent tasks to run in the wrong order. Also, what guarantees that vScheduler will be run after each task iteration is that the default FreeRTOS scheduling policy is priority based, and a ready task with a lower priority, which vScheduler is, will be guaranteed to run next.

Investigation:

FPS:

The principle behind this scheduling algorithm is that a ready task with a higher priority should run first. This leads to the situation when tasks with the lower priority never misses the deadline and tasks with lower priority, especially the lowest tasks, frequently lacks CPU time and misses their priority more frequently.

EDF:

The idea behind this scheduling mechanism is that the tasks that are closer to their deadline get CPU time. It makes processes execute without switching until completion or until they miss the deadline. In out case, Espresso will be called first, until it finishes its execution. Then Latte. And Cappuccino will miss its deadline. However it will be able to finish before the new period. Mocha will start brewing late but will be able to finish. With EDF we have one more deadline missed than with FPS.

LLP:

Least Laxity Priority should be more flexible in the sense of giving tasks enough CPU time. It uses special formula that calculates the laxity of a task. The process with the least laxity wins the race and gets CPU time. This also solves the problem of one task occupying CPU until completion. In our example, it appeared to be the most optimal. All tasks had enough CPU time and none of them missed a deadline. This is an amazing result as a simple formula allowed to distribute computing resource so that no process left behind.

SRTF:

In the SRTF scheduling algorithm, the least remining task gets executed first. As it lest shorter tasks to execute first, long tasks will suffer until the end and have a high risk of failing deadlines. On our opinion, although SRTF is useful in some real-time scenarios, it is not fair in case of our experiments. Experimental results have proven this assumptions.

```
During 30 ticks of FPS scheduler...
        Espresso has            0 missed deadlines
        Latte has               2 missed deadlines
        Cappuccino has          0 missed deadlines
        Mocha has               0 missed deadlines
```

```
During 30 ticks of EDF scheduler...
        Espresso has            0 missed deadlines
        Latte has               0 missed deadlines
        Cappuccino has          2 missed deadlines
        Mocha has               1 missed deadlines
```

```
During 31 ticks of LLF scheduler...
        Espresso has            0 missed deadlines
        Latte has               0 missed deadlines
        Cappuccino has          0 missed deadlines
        Mocha has               0 missed deadlines
```

```
During 31 ticks of LET scheduler...
        Espresso has            0 missed deadlines
        Latte has               0 missed deadlines
        Cappuccino has          2 missed deadlines
        Mocha has               1 missed deadlines
```

(LET stands for Short Remaining Time First)