Exercise set: 2
Tommaso Carraro
student number: 015147592

**Introduction to IoT: second set of exercises**

**First exercise**

1. I would use event-based programming because for the improvement of the quality and the safety of the driving experience this car uses only sensors. It is possible to implement these IoT functionalities as follow:
    a. when the GPS sensor detects a change in the speed limit information an handler is invoked by the scheduler and it imposes to the OS to show the new speed limit to the driver;
    b. every 12 hours the car sensors record data about the state of the vehicle. This event is managed by an handler that passes the data to the server. If the vehicle presents problems the SO will show useful informations on how to solve these problems in real time;
    c. queue management: it is possible to give a priority to the tasks explained (preemptive scheduling). For example, if the GPS event arrives together with the sensors record event it is possible to delay the sensors record event to allow the car to show the correct speed limit to the driver. In fact, even if sensors record event is delayed by some minutes it isn't a problem for the safety of the driver.

    Why not thread-based programming: since the functionalities of the vehicle are very limited compared to other IoT devices we can use event-based programming because it is easier to implement, in fact there aren't the memory management and the thread scheduling problem.
2. I would use preemptive scheduling: we need limited task time and the possibility to interrupt a task if it is necessary. As I said previously there could be two problems:
    a. the sensors record event arrives together with the speed limit information event: if the scheduler decides to execute the sensors record event before, we need to interrupt that event to allow to the speed limit information event to be managed. In fact, even if the sensors record event is delayed by some minutes it isn't a problem for the safety of the driver, but the speed limit information event can't be delayed because it is really important for the safety of the driver and other people;
    b. the sensors record event could require a lot of time to be completed: it could be possible that there isn't connectivity in a piece of road. In this case that event could require several minutes to be completed, so we need to be able to stop its management to give the other events the possibility to be managed.

    Usually preemptive scheduling has some problem with the battery management but in this case we can assume there aren't these types of problems because the battery capacity is very huge.
3. I would use Contiki OS instead of Android: since the functionalities of the IoT device are really limited we can avoid to use complex operating systems like Android. In fact, the functionalities of this device are comparable to the functionalities of a small

microcontroller and Contiki has been designed for this purpose. The benefits on the use of Contiki are:

   a. light operating system compare to Android: this car doesn't require applications and their ecosystem, so we can stay small in the kernel of the OS;
   b. lightweight preemptive thread on top of an event-based kernel: this is what we need in our car;
   c. built TCP/IP stack: we need the network layer to send sensors data to the server.

4. I would have firm constraints on real-time operation: it is true that the functionalities of the IoT device are important for the safety of the driver but they aren't critical for his life (a critical IoT functionality could be the autonomous driving). In fact, even if these events are delayed, it isn't a problem:
   a. if the car misses to set a new speed limit the driver should be able to check it from the road;
   b. if the sensors record event is delayed it isn't a problem because the prevoius check has been done 12 hours before and the algorithms didn't find problems in the state of the vehicle.

   It could not have a soft constraint simply because the functionalities are important for the safety of the driver and we can't delayed them forever.

**Second exercise - first application (enterprise IoT)**

1. The first application I have selected is smart trashing and the smart object is a big smart bin that have to be placed in a rubbish dump. The operators can put the rubbish inside the bin without sorting it because the main functionality of the bin is to sort the rubbish by itself. How does it work? When the rubbish is put inside the bin a camera is able to identify rubbish types: this is possible thanks to a complex computer vision algorithm that works behind the camera. After rubbish identification a robotic arm takes each item and puts it inside the correct bin. In fact, inside this big bin there are different smaller bins, one per rubbish type. Finally, through a touch screen it is possible to manage the big bin and to visualize useful information about its state. For example, when one of the smaller bin inside the smart object is full, the screen will visualize this information. It is possible to connect the smart object to the network of the rubbish dump in such a way that the bin can send notifications to the operators when something goes wrong.
2. This complex bin requires a real time operating system with hard constraint. In fact, the camera must be able to identify each item correctly and after that the arm should put it in the correct bin really fast because an interruption on the system could cause damages to the bin and delayes in the garbage sorting. To compute these real time operations fast and sofisticated algotithms in a efficienty way this device needs a thread based programming with a preemptive scheduling: each task requires its time and we can't go out of time because the bin is at the end of the chain and if an item takes forever to being sorted the all chain takes forever too. Moreover, we need to stop some tasks if there are problems: for example, if one of the bins is full we could need to stop the task that is sorting the garbage to execute the task that changes the bin. The bin requires communication support to be connected to the mobile platform

that sends useful information to operators' devices. The operating system should have real time capabilities to be able to sort each item in a fixed and strict time.

3. The best operating system for this device would be a real-time operating system, for example Nucleus RTOS as mentioned during lectures. We need this OS because the system is real-time and we are focused in its reliability and not in an app ecosystem.
4. The worst operating system for this device could be TinyOS simply because it is the simplest operating system designed for microcontroller and node sensors that have limited functionalities and resources and that's not the case, in fact it is unthinkable to run these complex real-time functionalities on TinyOS.

**Second exercise - second application (consumer level IoT)**

1. The second application I have selected is smart home and the smart object is a smart fridge. This fridge has a camera that keeps track of the food inside of it. Through this camera the fridge is able to learn what the user eat in a period of time in such a way that it can order the food by itself. Moreover, it is able to buy the food in an intelligent way: if a product is finishing or expiring the fridge will order the product. To be able to perform these checks it needs computer vision algorithms. With these functionalities the user doesn't need to go buy something or to check the fridge. It is possible to configure the fridge through a touch screen. With this trouch screen the user can disabilitate the machine learning algorithm that learns the foods in such a way that he can decide which types of food to order by himself. Then, it is possible to make subscription to retailer such as Amazon to organize food ordering and delivering. In conclusion, the fridge is connected to the network in such a way that it can send useful notifications about its state to the user's device and it can connect to other IoT devices such as oven or weight scale.
2. The operating system could be event-based because the fridge is all about sensors and touch screen events. For example, when a product is finishing the camera will launch an event that can be managed by an handler. This handler will impose to the fridge to order a new copy of the product. The scheduling could be non-preemptive, in fact there aren't tasks that are more important than other, so there isn't the necessity to have priorities. Since the fridge needs to order foods and to send notifications to the user's device it requires communication support.
3. I think the best suited operating system for this fridge is Tizen because it has all the functionalities needed to implement the software of the fridge. Tizen is widely used in smart home devices and its key design features are performance and programmability. Finally, it provides standard interfaces for integrating sensors data with web application development, so it could be easier to implement the functionalities compare to other operating systems.
4. The worst suited operating system could be again TinyOS. This OS has been designed for low-end devices with limited storage and that's not the case. This fridge needs resources and complex algorithms to perform its functionalities.

**Third exercise**

The script listens the messagges that are sent by the motes in the simulation and counts the number of occurrencies of 4 types of messagges. When every of these messagges' types have occurred for at least 5 times the script stops the simulation.