Ex02a

After some experimentation, we managed to get the following result:

‘Task name: hello Task name: hello Task name: hello Task name: hello Task name: hello’

The result comes either from the program running each task serial, or completing the print statement before doing a part of the another task. In the former case, it completes a task before starting the next. In the latter case, the program selects a random task, performs a line from it and then either continues with the next, or picks another task to run the next line of that one. Print statements are not interrupted, resulting in the ‘Task name: hello’ being written full without strange combinations (‘Tahelsk lo’, formed from combining ‘Task’ and ‘hello’).

Ex02b.

We had to replace in ‘rt\_printf("Task name: %s ", curtaskinfo.name);’ curtaskinfo.name with ‘num’ in order to acquire the following result:

‘Task name: 1 Task name: 2 Task name: 3 Task name: 4 Task name: 5’

This shows that both this exercise and previous exercise had the program do the tasks in serial: finish task 1 first, before starting task 2, task 3, etc.

Ex02c.

The order did not change as we swapped priorities. We have tried to swap the order of low to high-priority (20, 40,…, 80 to 80, 60, …, 20), made one priority very high (and discovering that the limit is on 99) and experiment with first creating all tasks before starting them, or creating a single task and starting them before doing the next. None managed to influence the order.

This might be for two reasons. Either the program first finishes a task for one-shot before starting the next one. In this case, priority does not matter: it simply finishes the task regardless of priority. An alternative is that the function is too short. By the time the program manages to start the next task, it has already managed to finish the first task.