

Design of Embedded Systems (DES), Assignment 2

Richard Bisschops: s4448545 & Lisa Boonstra: s3018547

Ex03a

We see that the first task counts from 1 to 10. After this the second task takes over and counts down till 0. The wanted behaviour is that the first task adds and the other task subtracts such that the output toggles between 0 and 1. However since the first task is started first and is not interrupted, it executes the for loop 10 times.

Ex03b

Our solution ensures that the global variable switches between 0 and 1, and can handle more tasks than two and is independent of the order of tasks started.

First, for the toggling between 0 and 1, we ensured that either an increasing task (increases global by 1) or a decreasing task (decreases global by 1) do not function at the same time. We use two semaphores for this. One semaphore represents the permission to increase the global by 1, the other the permission to decrease it. At the start, only one of the two semaphores allows a single task to work. This depends on whether the global is 0 (a single increasing task is allowed to work) or the global is 1 (a single decreasing task is allowed to work). Once an increaser task works, it will allow at the end of its execution for a single decreasing task to work by increasing the counter of the increasing semaphore by one.: it does not give permission to other increasing tasks to execute. Similarly, a decreasing task allows an increasing task to work at the end and not other decreasing task. Through this, only a single task can work, and it is always the opposite task.

For the bonus, we ensured that the order of when tasks are created does not matter by ensuring that all tasks are started at the same time. We use a third semaphore which closes off execution of any task, until a broadcast is used to wake every task up (after which the description of the above paragraph happens.) Further, the first type of task that can start is made dependent of the global: we either decrease the amount of permissions of increasing or decreasing the global by checking the global: if it is zero, no decreasing task is allowed to work first. Similarly, for when the global is one, the permission counter for increasing task is decreased by one.

We further ensured that multiple increasing and decreasing tasks work by ensuring that all tasks start at the same time using the third semaphore.

Ex03c

We created two files for each type: one for broadcast, and one for v. Broadcast works by waking up all tasks after each task is started. V works by letting all tasks wait till each is started, followed by increasing the permission of the semaphore by one. We discovered that for this type, we had to change the parameter of FIFO to PRIO, which was not necessary for Broadcast. Both broadcast and v only required one semaphore in order to have the priorities work as intended.

Ex04a

The task (task 2) that was given the highest priority was fully executed first. Task 1, which was given the second highest priority was executed fully afterwards, followed by execution of the last task which had the lowest priority (task0). This occurred because the priorities did not change during execution, thus the

task with the highest priority will be executed first.

Ex04b

The tasks are still executed fully. Task 1 ran first followed by execution of task 2. We expect that task 1 was executed first because this task was started first.

Ex04c

The second task has the highest priority at first. After it has passed its halfway point, of task execution, the priority of task 1 and task 0 are both increased by 10. The new priorities for each task are 61 (task 1) and 60 (task 0). Thus task 1 is executed fully after this halfway point, followed by the full execution of task 0. After task 1 and 0 are executed, the execution of task 2 resumes.

Ex04d

Task 2 runs until it has passed half-way its execution, after which the priority is changed to 38. Then task 1 runs, since it now has the highest priority. This task is also executed until half-way, after which the priority is set to 39. Task 0 runs after this, since it has priority (50), half-way through the execution the priority is set to 40. However as 40 is still the highest priority, task 0 executes fully first. After this task 1 has the highest priority and executes, followed by the execution of task 2.