Design of Embedded Systems (DES), Assignment 6

Richard Bisschops: s4448545 & Lisa Boonstra: s3018547

Ex06b

Our solution delivers the result as expected by the exercise. It works similar to the semaphore code, except the semaphore shared by high and low became a mutex. We had to add in a third semaphore to prevent the low priority task from finishing early, as a task created by the main thread apparently shares the lock and can skip the acquire part, even if the main thread has acquired the mutex. The "rt_sem_p(&lowhigh, (RTIME) TM_INFINITE); " line is replaced by "rt_mutex_acquire(&lowhigh, (RTIME) TM_INFINITE);" and "rt_sem_v(&lowhigh);" is replaced by "rt_mutex_release(&lowhigh);" in functions of the lowest and highest task.

As can be seen, once the high priority task tries to lock the mutex, the low priority task get a temporary higher priority to the middle task till it releases the lock: afterwards, priorities work again as intended. This is because once the high priority task tries to acquire the lock, it cannot do so. The mutex then takes the higher priority of the high priority task and gives it to the lower priority task until it finishes: afterwards, the low priority task gets it former priority back, causing the higher priority task to finish first, the medium second and the low priority task third.