Laboratory 2

Olafur Jonsson Konstantinos Xyderos

October 6, 2020

2.3 Questions

TODO

3.1 Accessing a Shared Resource: I/O

What is the expected output of this program in the first glance?

The program consists of 3 tasks. The first two simply echo "Hello from Task[number]" where [number] is the task number (1 or 2). The third task outputs information about stack space usage of each task. From looking at the code, an expected output would be:

```
Hello from Task1
Hello from Task2
Task1
```

The program might not show the desired behavior. Explain why this might happen.

While not completely unexpected, some output from a task might be mixed with output from a different task due to preemptiion from a task with a higher priority. The highest priority task (Task 1) should however not be preempted since it has the highest priority.

What will happen if you take away OSTimeDlyHMSM() statements? Why?

Task 1 will loop indefinitely and without being preempted by any other task since it has the highest priority. The program will simply output "Hello from Task1" in an infinite loop.

Semaphores can help you to get the desired program behavior. What are semaphores? How can one declare and create a semaphore in MicroC/OS-II?

Semaphores are a low level signal mechanism which provides a way to control access to resources. A integer variable which can be incremented and decremented in a thread-safe fashion is one way to describe a semaphore. µC/OS-IIhas built in functions for creating and managing semaphores. By initializing an `OS_EVENT` structure using `OSSemCreate()`, a semaphore is created which can be used with various functions such as `OSSemPend()`, `OSSemPost()` among other (µC/OS-IIAPI Reference).

How can a semaphore protect a critical section? Give a code example!

```
// declare the semaphore as global
OS_EVENT * criticalSemaphore;

// in main thread...
void main() {
    // create and initialize the semaphore with an initial value of 1 (not busy)
    criticalSemaphore = OSSemCreate(1);

    // ...
}

// in task code:
void task() {
    // wait until criticalSemaphore has a value of 1
    OSSemPend(criticalSemaphore, 0, &err);

    // do critical stuff
    criticalstuff();

    // signal the semaphore, indicating that the critical part is done
    // some other task waiting on OSSemPend() can now continue
    OSSemPost(task1StateSemaphore);
}
```

Who is allowed to lock and release semaphores? Can a task release a semaphore that was locked by another task?

There are no safeguards in µC/OS-IIto prevent unwanted lock and unlocking behaviour of individual tasks, so it's up to the programmer to make sure not to call `OSSemPost()` without a preceding `OSSemPend()` for example.

Explain the mechanisms behind the command OSSemPost() and OSSemPend()!

TODO

Draw the new application as a block diagram containing processes, semaphores and shared resources. Use the graphical notation which has been used in the lectures and exercises for this purpose, and included in Figure 4 from Appendix A.

TODO