

### Slide 1

Look now at program in file example4.c, which is an example of signaling applications using binary semaphores. Task 3 runs the shown function, which repeats its operation after a delay of 1 second. The function waits for two signals by performing a semaphore take twice, before displaying a message indicating that it was unblocked.

### Slide 2

The tasks 1 and 2 give the semaphore each second, and each displays a string when it runs.

### Slide 3

Here, we disable task 2 and only tasks 1 and 3 are running. As expected, task 3 will be unblocked after task 1 iterates two times, as indicated by the output displayed.

### Slide 4

Now we run the three tasks at the same priority level. We observe that task 3 runs only after each of tasks 1 and 2 run two times, not one time as may first be expected. Try to explain this before going to the next slide.

### Slide 5

#### Example 4

Since the semaphore used is a binary semaphore, its value cannot exceed 1. When task 1 first gives the semaphore, its value becomes 1. If task 2 performs next the "Give" operation, it fails. It cannot be successful until task 3 performs one take operation first, then the Give operation has to be repeated.

To have task 3 running after each of tasks 1 and 2 run once, any of the following can be done:

- Use a counting semaphore, so two successive gives succeed.
- Use two binary semaphores, one for each task to give.
- Give task 3 higher priority, so it takes the semaphore given by task 1 before task 2 gives it again.

These ideas are tested in the next slides.

ECP-622- Spring 2020

### Slide 6

Here, we changed the semaphore into a counting semaphore.

### Slide 7

Here, we use two different binary semaphores. Each of tasks 1 and 2 gives a different semaphore. Task 3 then takes the two semaphores before running.

### Slide 8

Here, we run the original program, but give task 3 a higher priority.