Synchronization with Semaphores

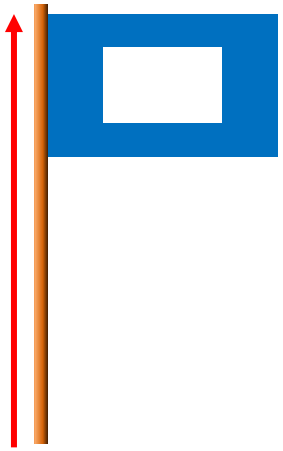
Event Flag Limitations

Switch		Press		Release	Press	Release	Press	Release			
ISR		FlagSet			FlagSet		FlagSet				
Flag	0	1	0	0	1	1	1	1	1	0	0
Thread	FlagWait		Running						FlagWait	Running	

- What if we press the switch three times during the first flash sequence?
  - We will only get two scans, not three
  - We lose one event/trigger
  - Flags only count up to 1, can't track more pending events
- Also: No handshaking
  - OK to set a flag which is already set.
  - Does sender know receiver got notification?
    - No, unless it checks with extra code….
- Also: No data included, just notification that the event occurred

Semaphore Basics

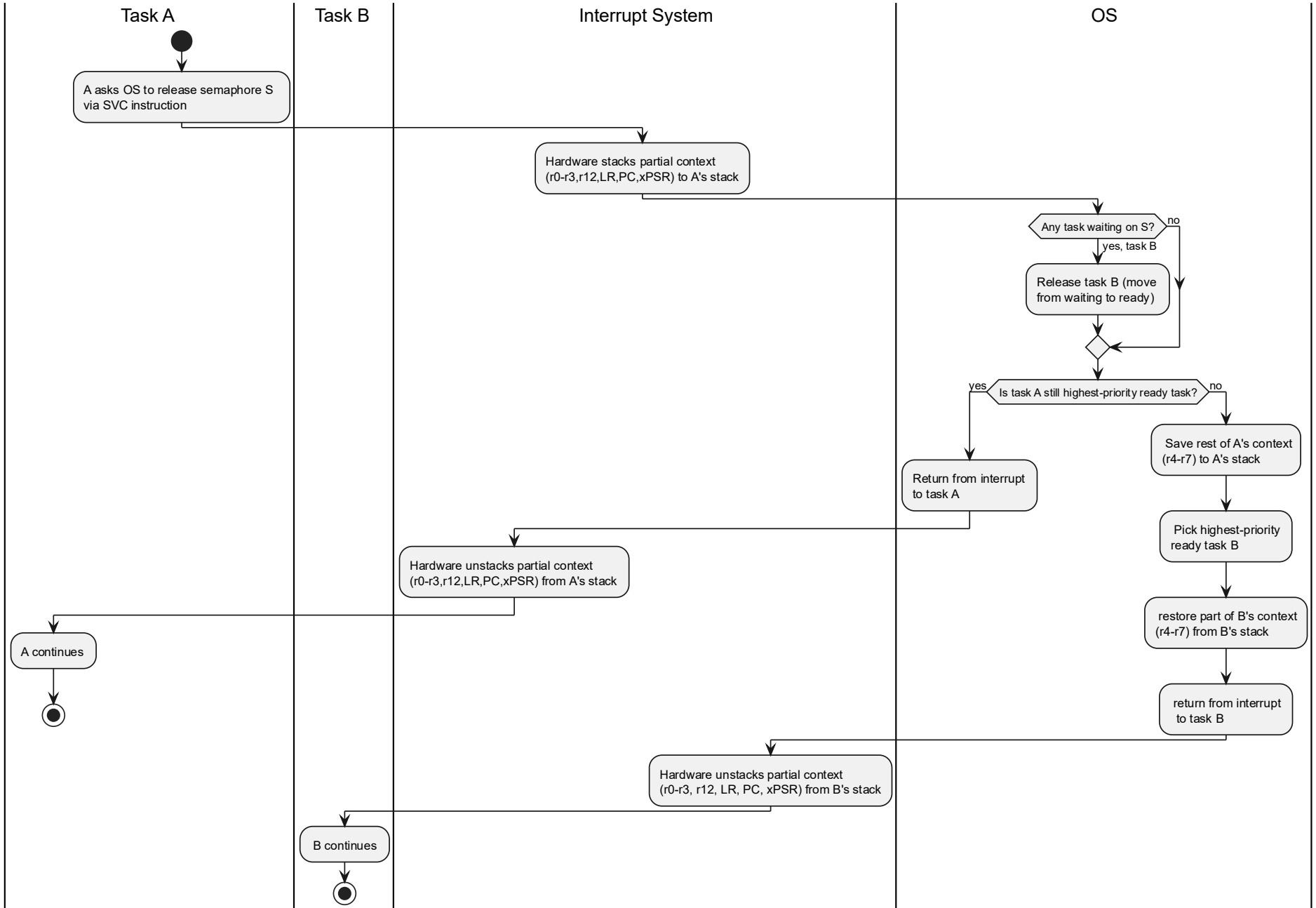
1. Semaphore not raised yet (count == 0)



2. Thread A releases semaphore, count is incremented

3. When Thread B successfully acquires semaphore, count is decremented

- Semaphore: special OS-managed *counter* variable.
  - Access it **only** using special OS calls
- Value of semaphore counter indicates how many times it has been released without being acquired
  - Can only acquire semaphore if *count* > 0
- osSemaphoreRelease
  - Increments *count.* Calling thread is ready to run.
- osSemaphoreAcquire
  - *count* > 0? Succeeds! OS decrements *count* and calling thread is ready to run.
  - *count* == 0? Fails! OS puts calling thread in blocking state, waiting for semaphore *count* > 0


Task A
Task B
Interrupt System
OS
A asks OS to release semaphore S via SVC instruction
Hardware stacks partial context (r0-r3,r12,LR,PC,xPSR) to A's stack
Any task waiting on S?
no
yes, task B
Release task B (move from waiting to ready)
yes
Is task A still highest-priority ready task?
no
Return from interrupt to task A
Save rest of A's context (r4-r7) to A's stack
Pick highest-priority ready task B
restore part of B's context (r4-r7) from B's stack
return from interrupt to task B
Hardware unstacks partial context (r0-r3,r12,LR,PC,xPSR) from A's stack
A continues
Hardware unstacks partial context (r0-r3, r12, LR, PC, xPSR) from B's stack
B continues

A Signals B Using Semaphore

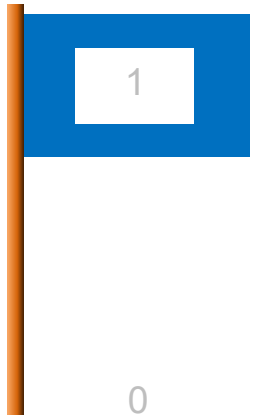
- Thread A releases the semaphore
- Resulting OS actions
  - If any task is waiting for this semaphore, move that task to *ready* state, give it the semaphore.
  - Run highest-priority ready task.

- Thread B tries to acquire semaphore
- Resulting OS actions
  - If semaphore is free, then give it to B. Else put B into *waiting/blocking* state.
  - Run highest-priority ready task.

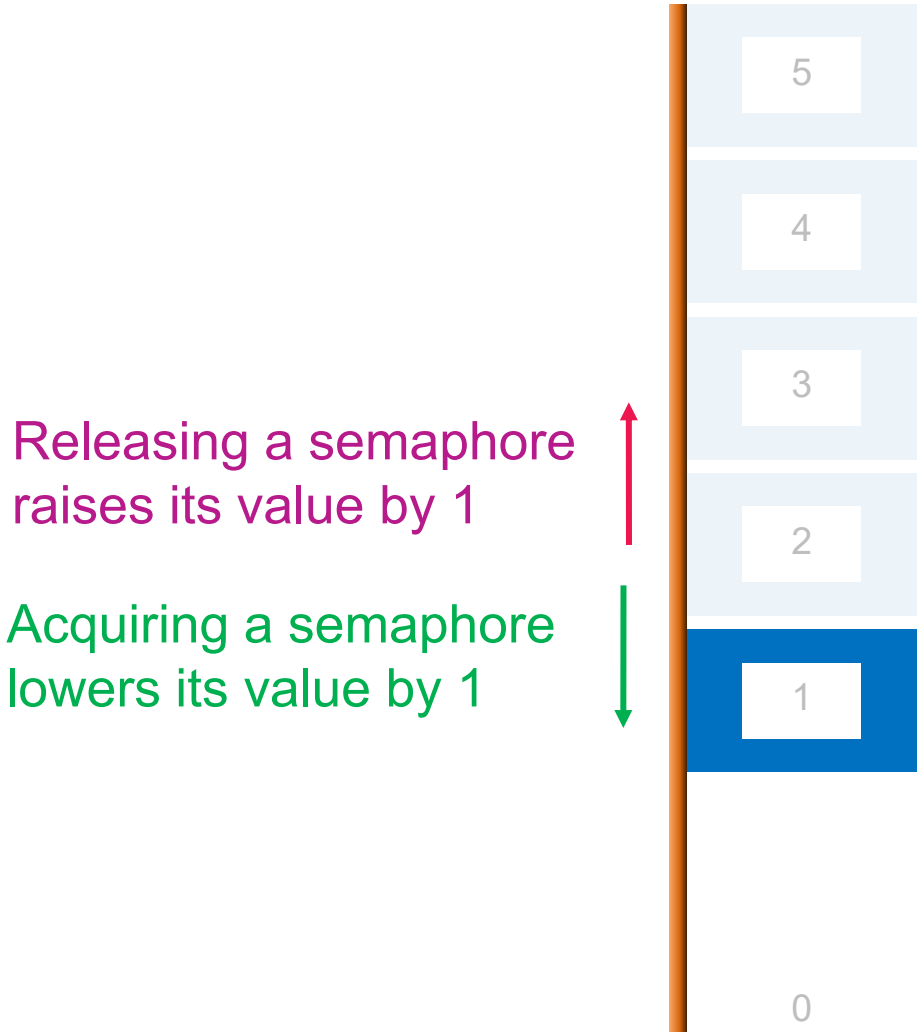
Counting vs. Binary Semaphore

How tall is the flagpole?

- If we need multiple pending events to be recognized, then use a **counting** semaphore
  - Accumulates number of pending requests (often called *tokens*)
- **Binary** semaphore only goes up to 1 (like event flag)
- Define using *max_count* parameter in osSemaphoreNew call
  - Demo example: switch USE_COUNTING_SEM between 0 and 1




```
RGB_sem =
osSemaphoreNew(1,0,NULL);
```



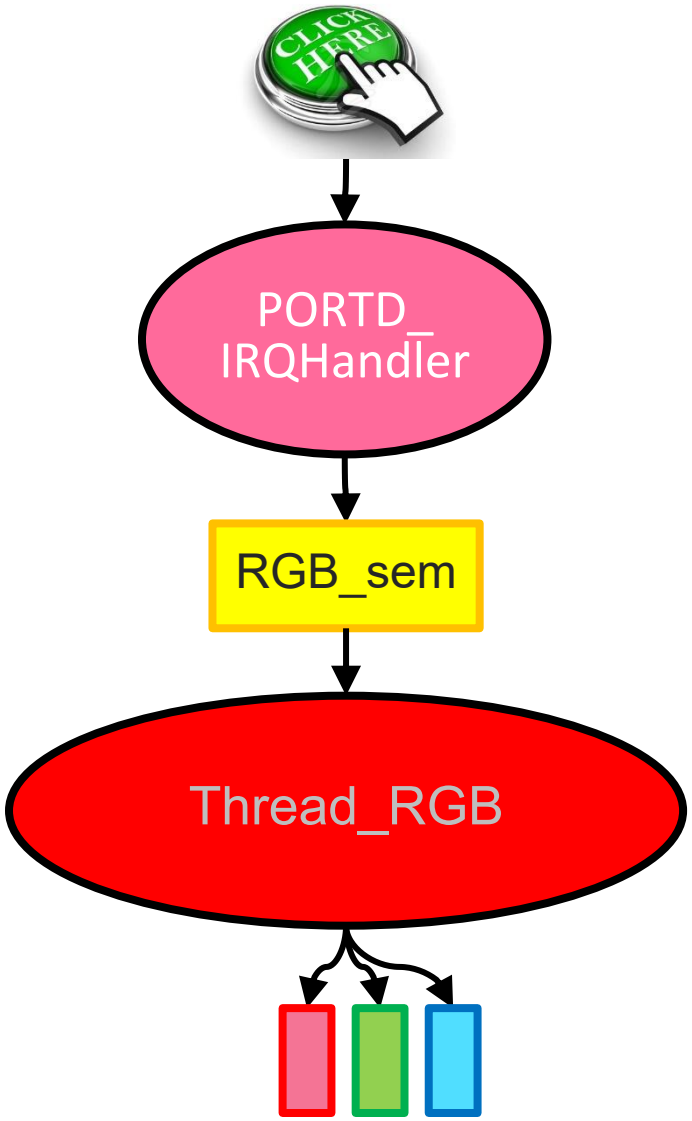

```
RGB_sem =
osSemaphoreNew(5,0,NULL);
```

CMSIS-RTOS2 Semaphores

- Type: osSemaphoreId_t
- osSemaphoreId_t osSemaphoreNew(uint32_t max_count, uint32_t initial_count, const osSemaphoreAttr_t *attr)
  - Creates semaphore specified counts
  - Initializes it to initial_count
- osStatus_t osSemaphoreRelease(osSemaphoreId semaphore_id)
  - Raises the semaphore, incrementing count value
  - Result code indicates any error
    - osOK: token correctly released, count increased.
    - osErrorResource: maximum token count reached.
    - osErrorParameter: parameter incorrect.
- osStatus_t osSemaphoreAcquire(osSemaphoreId semaphore_id, uint32_t timeout)
  - Wait for semaphore to be signaled (if not already), then decrements count value
  - Optional timeout value, measured in kernel ticks
    - To never timeout, use osWaitForever
    - To never wait, use 0
  - Result code indicates result:
    - osOK
    - osErrorTimeout: timed out, didn't get semaphore
    - osErrorResource: didn't wait, didn't get semaphore

RTX5 Demo Semaphores: Semaphore for LED RGB Sequence

- Requirement: Light LEDs in RGB sequence **once** when button is pressed (leading edge of press)
- Use semaphore RGB_sem to indicate **request** for LED sequence
- PORTD_IRQHandler **releases (gives)** the semaphore once each time button is pressed
- Thread_RGB runs once each time it can **acquire (take)** the semaphore




```
init() {
  …
  RGB_sem = osSemaphoreNew(5,0,
                NULL);
}

PORTD_IRQHandler() {
  …
  osSemaphoreRelease(RGB_sem);
  …
}

Thread_RGB() {
  …
  osSemaphoreAcquire(RGB_sem,
  osWaitForever);
  …
}
```

CMSIS-RTOS2 Status and Error Codes

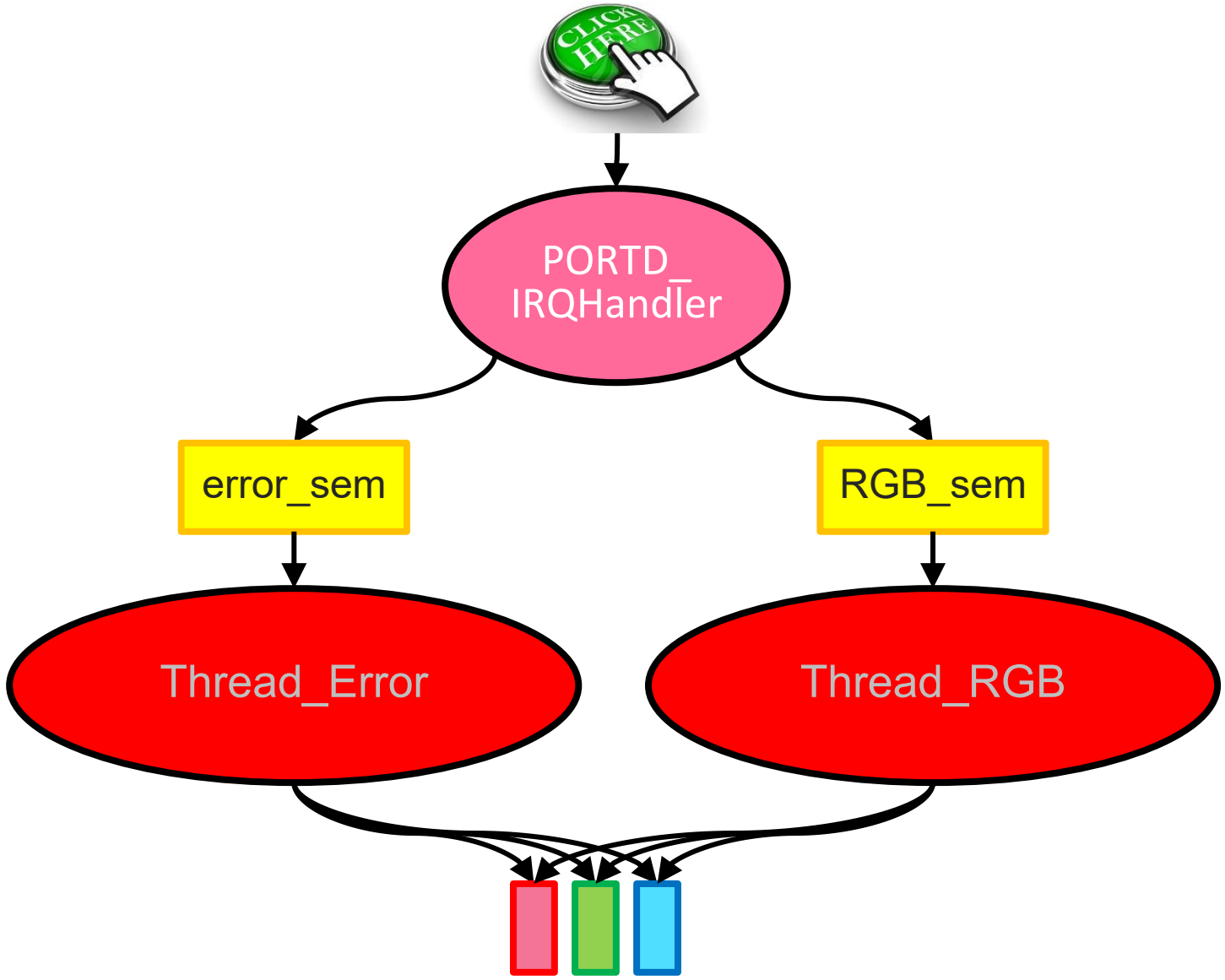
enum osStatus_t { … }

osOK	function completed; no error or event occurred.
osErrorParameter	mandatory parameter missing, incorrect object.
osErrorResource	resource not available
osErrorTimeout	operation not completed within the timeout period
osErrorISR	function cannot be called from interrupt service routines.
osErrorOS	unspecified RTOS run-time error.
os_status_reserved	prevent from enum down-size compiler optimization.

- A call to osSemaphoreRelease will fail if *counter* is at max_count
  - Yes, we should check the return value from osSemaphoreRelease

RTX5 Demo Semaphores: Error Handling

- Notify of RGB_sem release error with another semaphore error_sem
  - Could also use event flag. Discuss.
- Thread_Error blocks on error_sem. If acquired, thread flashes red LED forever
- Enable by defining USE_ERROR_HANDLING as non-zero (e.g. 1)



Event Flags vs. Semaphores

- Unlike event flags and thread event flags, can't simultaneously wait on AND/OR combinations of multiple semaphores

- Note
  - Semaphores and event flags can't send data.
  - They just indicate that *the event* occurred

- More on flags vs. semaphores
  - http://info.quadros.com/blog/rtos-explained-understanding-event-flags/
  - http://ecos.sourceware.org/docs-latest/ref/kernel-flags.html
  - http://ecos.sourceware.org/docs-latest/ref/kernel-semaphores.html