Semaphore in RTEMS

Kuan-Hsun Chen

LS 12, TU Dortmund

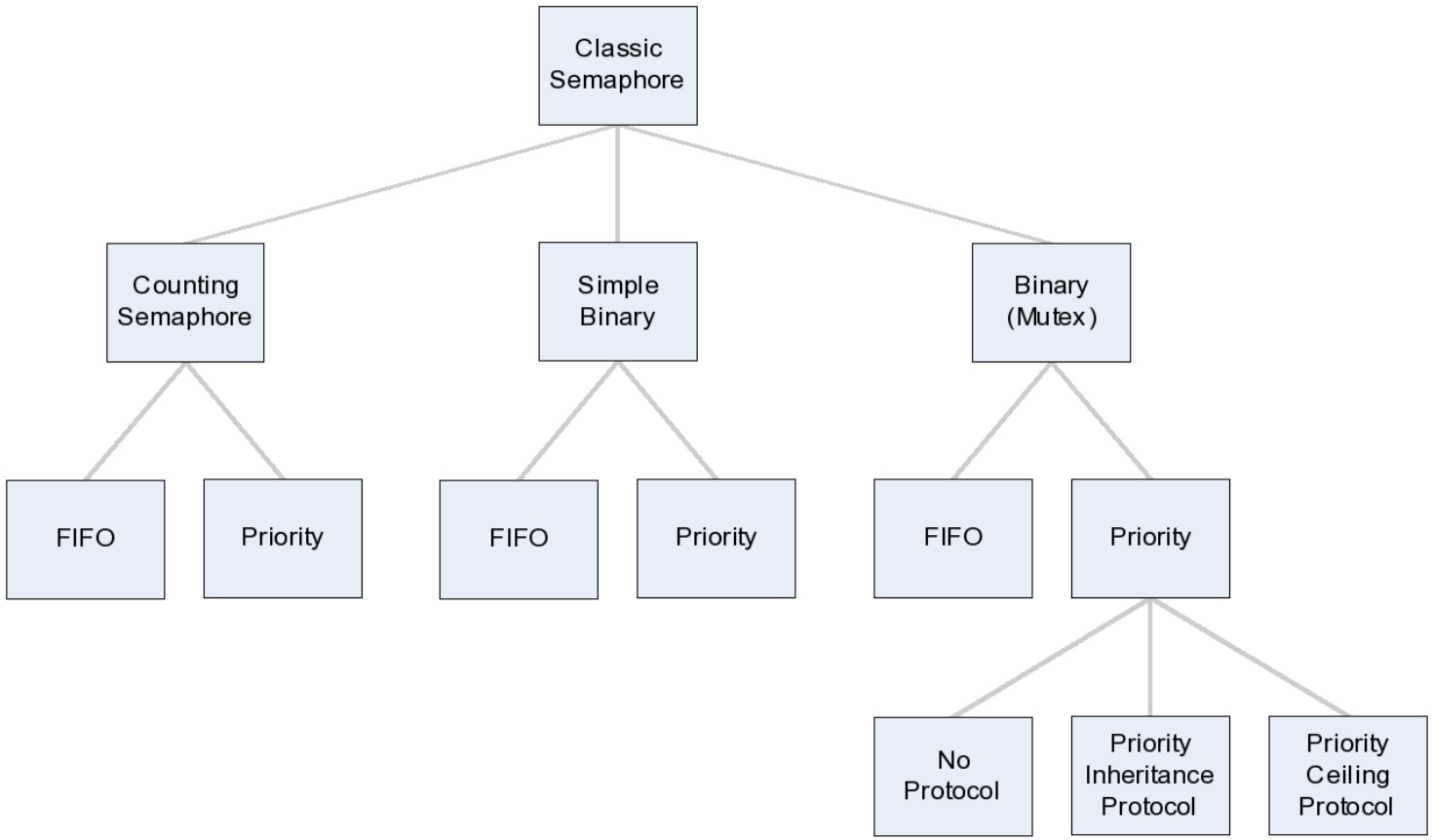
04,08,2015

Outline

- Introduction of Semaphore in RTEMS
- Priority Inversion
- Priority Inheritance Protocols
- Exercises

Semaphore in RTEMS

- Semaphore Attribute Set (Possible combinations)



- Source from `https://docs.rtems.org/doc-current/share/rtems/html/c_user/Semaphore-Manager-Building-a-Semaphore-Attribute-Set.html`

Features of RTEMS

- rtems_semaphore_create(name, count, attribute_set, priority_ceiling, rtems_id *id)
- Some attributes:
  - RTEMS_FIFO - tasks wait by FIFO (default)
  - RTEMS_PRIORITY - tasks wait by priority
  - RTEMS_COUNTING_SEMAPHORE - no restriction on values (default)
  - RTEMS_BINARY_SEMAPHORE - restrict values to 0 and 1
  - RTEMS_NO_INHERIT_PRIORITY - do not use priority inheritance (default)
  - RTEMS_NO_PRIORITY_CEILING - do not use priority ceiling (default)
  - RTEMS_LOCAL - local semaphore (default)
  - ...
- For example: RTEMS_BINARY_SEMAPHORE | RTEMS_FIFO | RTEMS_NO_INHERIT_PRIORITY | RTEMS_NO_PRIORITY_CEILING | RTEMS_LOCAL
- Count should be larger than 1 for the normal usage of binary/counting semaphore.

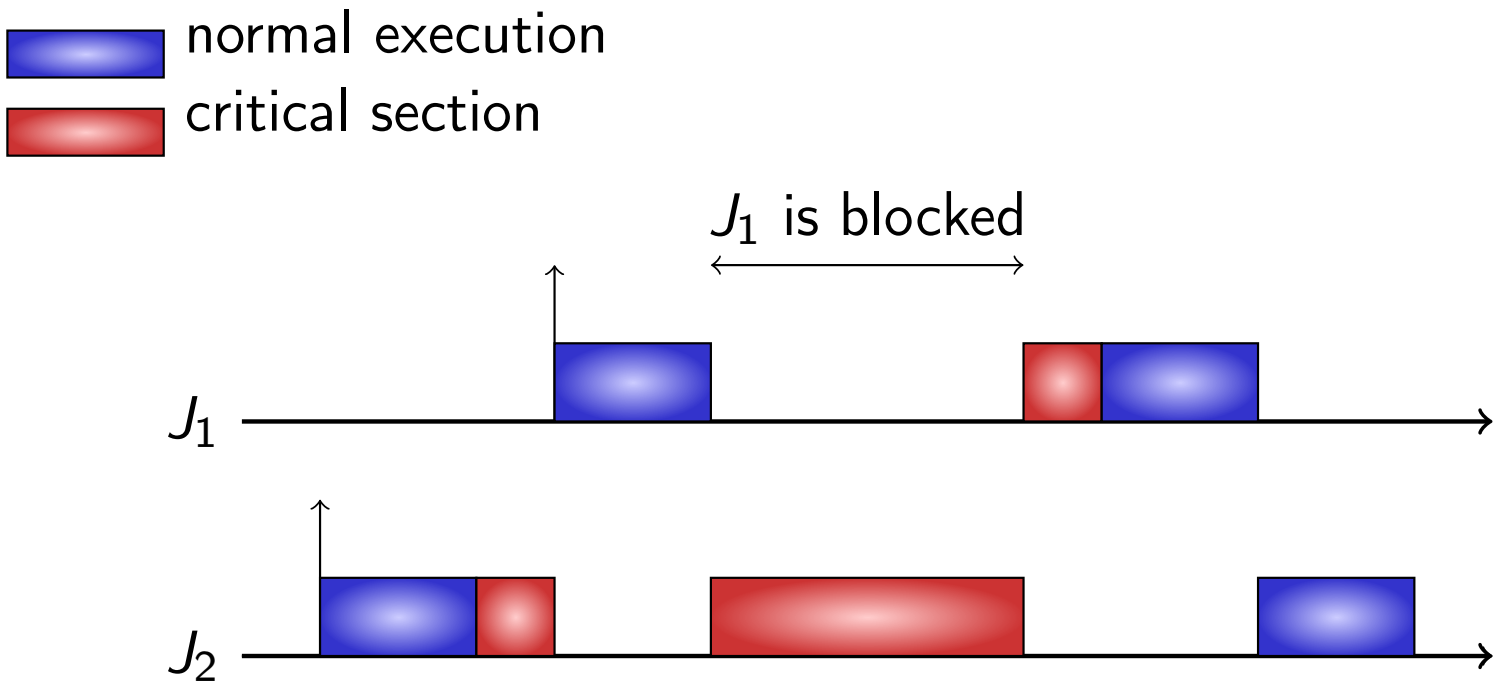
Dig into the source code cpukit/

- In fact Binary Semaphore in RTEMS is implemented by the structure of Mutex. (Binary Semaphore != Mutex)
- RTEMS interface:
  - rtems_semaphore_obtain() is implemented into rtems/src/semobtain.c
  - rtems_semaphore_release() is implemented into rtems/src/semrelease.c
- Core functions:
  - _CORE_mutex_Seize_interrupt_blocking() in score/src/coremutexseize.c
  - _Thread_Raise_priority() in score/src/threadchangepriority.c
  - inline functions in score/include/rtems/score/.

Priority Inversion

A higher priority job is *blocked* by a lower-priority job.

- Unavoidable when there are critical sections



Priority Inheritance Protocol (PIP)

When a lower-priority job J_j blocks a higher-priority job, the priority of job J_j is *promoted* to the priority level of highest-priority job that job J_j blocks.

For example, if the priority order is $J_1 > J_2 > J_3 > J_4 > J_5$,

- When job J_4 blocks jobs J_2 and J_3, the priority of J_4 is promoted to the priority level of J_2.
- When job J_5 blocks jobs J_1 and J_3, the priority of J_5 is promoted to the priority level of J_1.

Exercises (10 points)

1. Please build the source code and execute SEMAPHORE_TEST example. Then, draw the diagram to check the system behaviours. (3 points)
2. Remove the marked thread_raise_priority() in coremutexseize.c to recover PIP behaviours and draw the timing diagram. (2 points)
3. Revise PIP to promote the priority of resource holder to the highest priority. What is the drawback? Please explain in the diagram. (5 points)
   Hint: coremutexseize.c, taskcreate.c, grep -r "XXX" *