Real-Time Operating Systems

ROS01

Minor Embedded Systems

Week 7
RT Analyses (part 2)

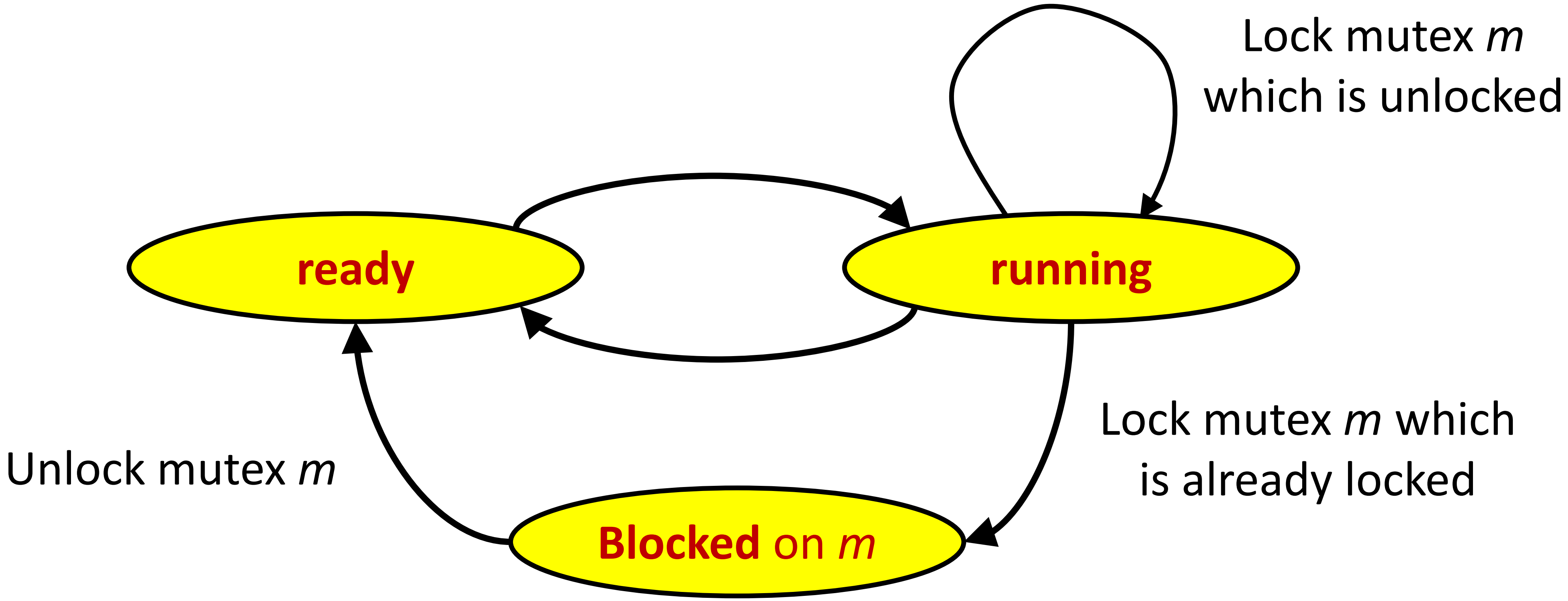
HOGESCHOOL ROTTERDAM

versd@hr.nl
brojz@hr.nl

Planning ROS01

- Week 1: Introduction – Blinking leds
- Week 2: Super loop construct with an ISR
- Week 3: Cooperative Scheduling
- Week 4: Pre-emptive Scheduling
- Week 5: Using TI-RTOS
- Week 6: Schedulability Analyses, Priority Assignment, Response Time Analyses (part 1)
- **Week 7: Response Time Analyses (part 2)**
- Week 8: Finalizing Final Assignment

Task states



Lock mutex *m*
which is unlocked
ready
running
Unlock mutex *m*
Lock mutex *m* which
is already locked
Blocked on *m*

FPS-DMPO Blocking

- When a task with a lower priority has to wait on a task with a higher priority, the task is preempted.
- A preempted task is added to the ready queue before tasks with the same priority.
- When a task with a high priority has to wait on a task with a lower priority, the task is blocked (priority inversion).
- When a task is unblocked, it is added to the ready queue after tasks with the same priority.
- To predict the real-time behavior of a task, the maximum time a task can be blocked must be predictable (bound blocking).

Priority inversion example

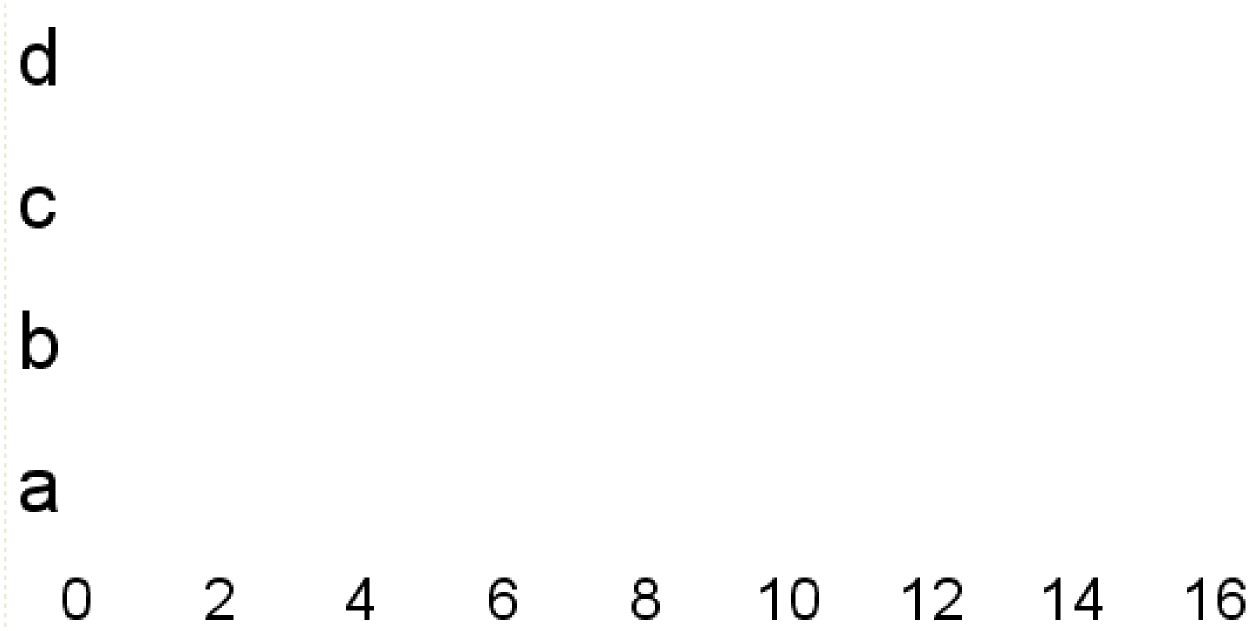
- Four tasks (a, b, c, and d) share two resources (Q and V).
- Each resource can only be used mutually exclusive (so each resource is protected with a mutex).

task	prio	execution	release time
d	4	EEQVE	4
c	3	EVVE	2
b	2	EE	2
a	1	EQQQE	0

E = task only needs the processor to run
Q = task needs processor and resource Q to run
V = task needs processor and resource V to run

Priority inversion example

Finish this **Gantt chart** yourself!



- Executing (yellow)
- Executing with Q locked (blue)
- Executing with V locked (cyan)
- Preempted (white)
- Blocked (red)

task	prio	execution	release time
d	4	EEQVE	4
c	3	EVVE	2
b	2	EE	2
a	1	EQQQE	0

FPS-DMPO Priority inversion

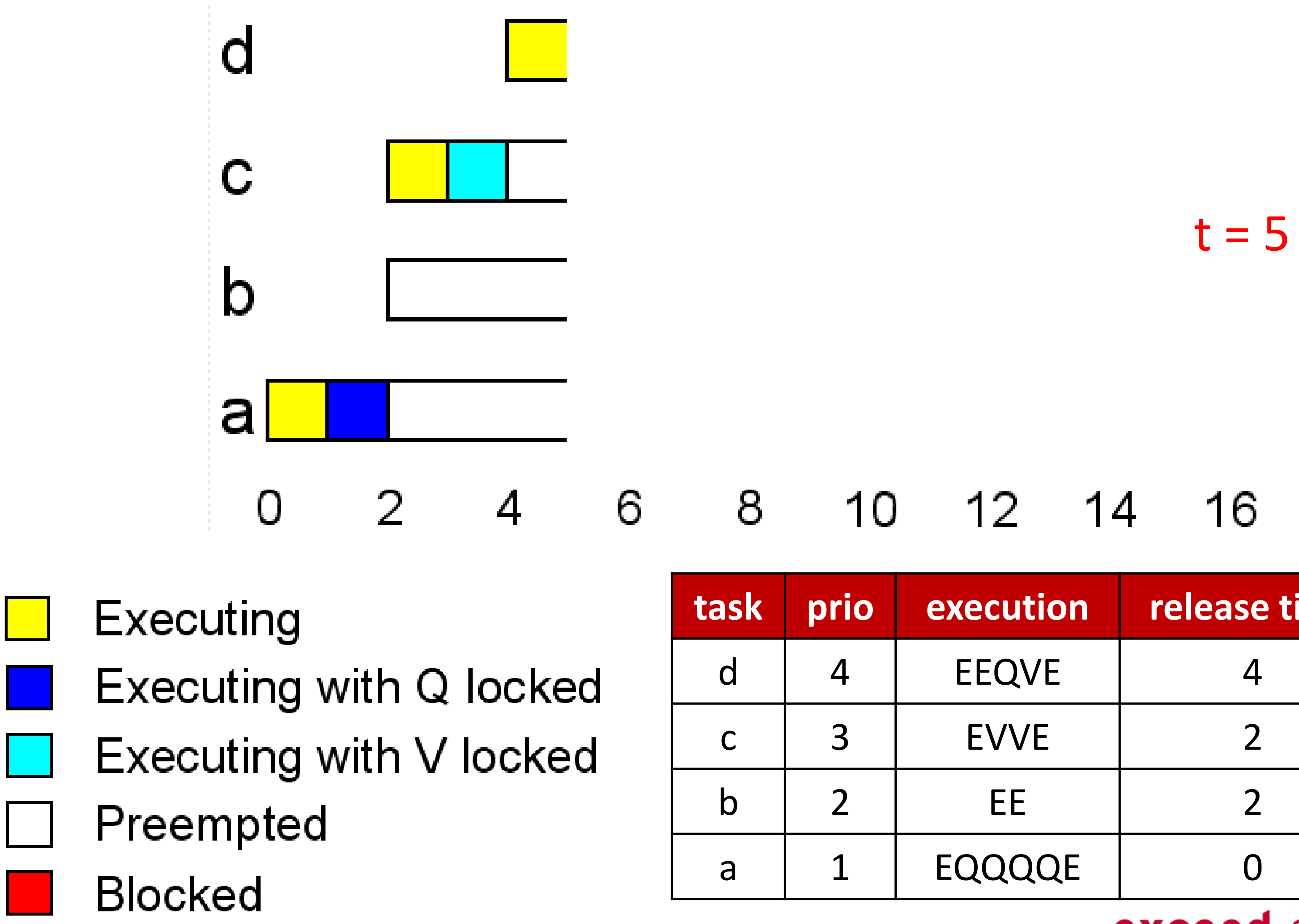
- Task d is being blocked by task a, b, and c (all tasks with a lower priority)!

- Blocking (priority inversion) **can not** be **avoided** if we use mutual exclusive recourses.

- Blocking **can** be **bounded** by using priority inheritance:
  - When a task is blocked on a resource, then the task that owns the recourse gets (inherits) the priority of the blocked task.

Priority inheritance example

Finish this **Gantt chart** yourself!



task	prio	execution	release time
d	4	EEQVE	4
c	3	EVVE	2
b	2	EE	2
a	1	EQQQE	0

Blocking Priority inheritance

- The blocked time of each task is now bounded.

$$B_i = \sum_{k=1}^{K} usage(k,i)C_k$$

- $\boldsymbol{B_i}$ = maximum blocking time for task $\boldsymbol{i}$
- $\boldsymbol{K}$ = total number of resources
- ***usage(k, i)*** = Boolean function
  - 1 if there is a task with a priority lower than $\boldsymbol{P_i}$ and a task with a priority higher than or equal to $\boldsymbol{P_i}$ (this can be task $\boldsymbol{i}$ itself) which share resource $\boldsymbol{k}$.
  - 0 otherwise.
- $\boldsymbol{C_k}$ = maximum time for which resource $\boldsymbol{k}$ is locked.

Blocking Response time analyze

$$R_i = C_i + B_i + I_i$$

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

Priority inheritance example

- Calculate the maximum blocking time ($\boldsymbol{B_i}$) for all tasks in the previous example

$$B_i = \sum_{k=1}^{K} usage(k,i) C_k$$

task	prio	execution	release time
d	4	EEQVE	4
c	3	EVVE	2
b	2	EE	2
a	1	EQQQE	0

E = task only needs the processor to run
Q = task needs processor and resource Q to run
V = task needs processor and resource V to run

usage(k, i) = 1 if there is a task with a priority lower than $\boldsymbol{P_i}$ and a task with a priority higher than or equal to $\boldsymbol{P_i}$ (this can be task ***i*** itself) which share resource ***k***.

Solution

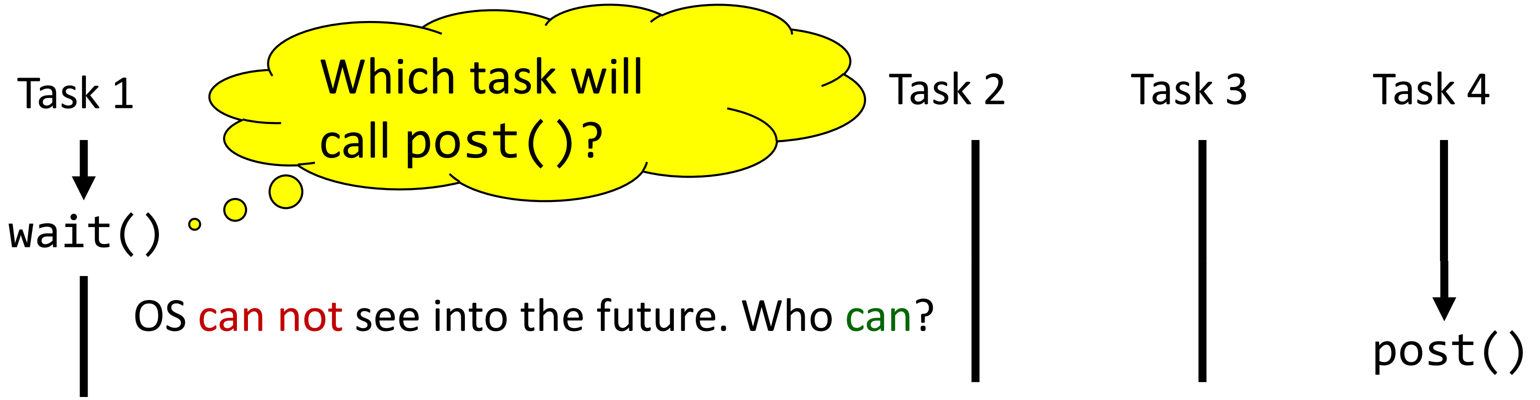
$C_V = 2, C_Q = 4$

Table for $usage(k, i)$ and B_i

i	k = V	k = Q	B_i
d	1	1	6
c	0	1	4
b	0	1	4
a	0	0	0

Blocking Priority inheritance

- Priority inheritance can not be implemented for semaphores and message queues!
  - When using a semaphore it is often not possible to determine which task is causing the blocking (which task will call the `post()` for which a task blocked on `wait()` is waiting for)!
  - Example: using a semaphore with an initial count value of zero for synchronization purposes.



  - When using message passing it is often not possible to determine which task is causing the blocking (which task will perform the `send()` for which a task blocked on `receive()` is waiting for)!
- Solution: e.g. Priority Ceiling Protocol