FPS-RMPA

$D<T$ en Sporadic taken

- $D<T$:
  - Gebruik DMPA in plaats van RMPA (geeft DMPO):

    $D_i < D_j \Rightarrow P_i > P_j$
  - Gebruik bij response tijd analyse als stop voorwaarde:

    $W_i^{n+1} > D_i$
- Sporadic taken:
  - Vul voor de periodetijd de minimale tijd tussen twee “starts” van deze taak in. T = minimum inter-arrival interval.
  - Vaak geldt voor sporadic taken $D<T$

THRijswijk

FPS-DMPO

Blocking

- Als een taak met een hoge prioriteit moet wachten op een taak met een lagere prioriteit dan wordt deze taak **blocked** (priority inversion).
- Een taak die unblocked sluit achter de taken met dezelfde prioriteit in de readyqueue aan.
- Als een taak met een lagere prioriteit moet wachten op een taak met een hogere prioriteit dan wordt deze taak **preempted**.
- Een taak die preempted wordt sluit voor de taken met dezelfde prioriteit in de readyqueue aan = vooraan de readyqueue!
- Om het real-time gedrag te kunnen voorspellen moet de maximale tijd dat een taak blocked kan zijn voor-spelbaar zijn (bound blocking).

THRijswijk

Voorbeeld

Priority inversion

- 4 taken (a, b, c en d) delen 2 resources (Q en V).
- Deze resources kunnen slechts door 1 taak tegelijk gebruikt worden (en zijn dus beschermd met bijvoorbeeld een mutex).

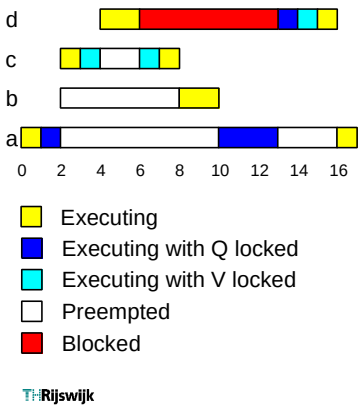
taak	prio	execution	release time
d	4	EEQVE	4
c	3	EVVE	2
b	2	EE	2
a	1	EQQQQE	0

  - E = taak heeft alleen processor nodig
  - Q = taak heeft resource Q nodig
  - V = taak heeft resource V nodig

THRijswijk

Voorbeeld

Priority inversion



THRijswijk

FPS-DMPO

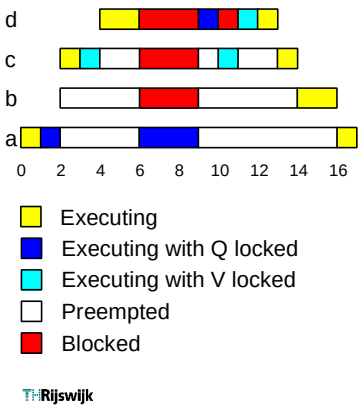
Priority inversion

- Taak d wordt blocked door taak a, b en c (alle processen met een lagere prioriteit)!
- Blocking (priority inversion) is niet te voorkomen als we mutual exclusive recources delen.
- Blocking is wel te beperken door het toepassen van **priority inheritance**:
  - Als een taak blocked voor een resource dan krijgt de taak die de resource bezit de prioriteit van de taak die blocked wordt.

THRijswijk

Voorbeeld

Priority inheritance



THRijswijk

Blocking

Priority inheritance

- De tijd dat een taak blocked kan zijn is nu beperkt.

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

- B = maximum blocking time
- K = aantal gebruikte resources
- $usage(k, i)$ = boolean functie
  - true als er een taak is met prioriteit lager dan P_i en een taak met prioriteit hoger of gelijk aan P_i die de resource k delen.
- $C(k)$ = maximale tijd dat resource k gebruikt wordt.

THRijswijk

Blocking

Response time analyse

$$R_i = C_i + B_i + I_i$$

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

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

THRijswijk

Blocking

Priority inheritance

- Priority inheritance kan **niet** eenvoudig geïmplementeerd worden!
  - Bij semaphoren en condition variabelen is vaak niet te achterhalen op wie je wacht (wie de post of signal zal geven)!

Taak 1 → sem_wait

Taak 2 → sem_post

  - Bij message passing is het vaak niet te achterhalen op wie je wacht.
- Alternatief: **priority ceiling**

THRijswijk

FPS-DMPO

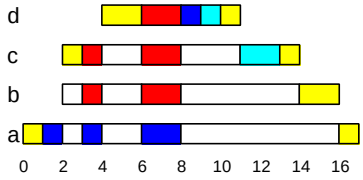
Priority ceiling

- Original Ceiling Priority Protocol OCPP:
  - Elke taak heeft een static priority.
  - Elke resource heeft een ceiling priority = maximale prioriteit van de processen die deze resource delen.
  - Een taak die een taak met een hogere prioriteit blocked erft de hogere prioriteit.
  - Een taak kan alleen een resource gebruiken als zijn prioriteit hoger is dan de ceiling van alle door andere taken gebruikte resources.
- Voordelen:
  - proces kan maar 1x blocked worden.
  - deadlock is niet mogelijk.
  - mutual exclusive “van zelf” goed.
  - transitive blocking is niet mogelijk.

THRijswijk

Voorbeeld

OCPP



ceiling Q = 4, ceiling V = 4

- Executing
- Executing with Q locked
- Executing with V locked
- Preempted
- Blocked

THRijswijk

Blocking

OCPP

- Eerste resource kan altijd worden gebruikt.
- Tweede resource kan alleen worden gebruikt als er geen taak is met een hogere prioriteit die **beide** resources gebruikt.
- De tijd dat een taak blocked kan zijn wordt nu:

$$B_i = \max_{k=1}^{K} usage(k,i)C(k)$$

- Impementatie is nog steeds net zo moeilijk!
- Alternatief: **Immediate CPP**.

THRijswijk

FPS-DMPO

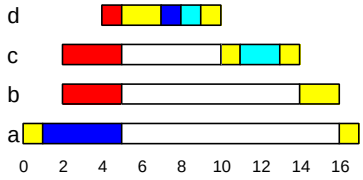
Priority ceiling

- Immediate Ceiling Priority Protocol ICPP:
  - Elke taak heeft een static priority.
  - Elke resource heeft een ceiling priority = maximale prioriteit van de processen die deze resource delen.
  - Als een taak een resource gebruikt krijgt die taak de ceiling prioriteit van de resource.
- Voordelen:
  - proces kan maar 1x blocked worden.
  - deadlock is niet mogelijk.
  - mutual exclusive “van zelf” goed.
  - transitive blocking is niet mogelijk.
  - ten opzichte van OCPP.
    - eenvoudiger te implementeren.
    - minder taak wisselingen.

THRijswijk

Voorbeeld

ICPP



ceiling Q = 4, ceiling V = 4

- Executing
- Executing with Q locked
- Executing with V locked
- Preempted
- Blocked

THRijswijk

POSIX

Priority Based Scheduling

- FIFO
  - Een thread blijft running totdat de thread blocked of preempted wordt.
  - Een thread die preempted wordt komt **vooraan** de readyqueue te staan.
- Round-Robbin
  - Een thread blijft running totdat de thread blocked of preempted wordt of totdat de time-slice is verstreken.
  - Een thread die preempted wordt komt **vooraan** de readyqueue te staan.
  - Een thread waarvan de time-slice is verstreken komt **achter** de threads met gelijke prioriteit op de readyqueue te staan.
- Sporadic Server
- Other

THRijswijk

IPPC

Priority Protect Protocol

- int pthread_mutexattr_getprotocol(const pthread_mutexattr_t* attr, int* protocol);
- int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr, int protocol);
  - mogelijke waarden voor protocol:
    - PTHREAD_PRIO_NONE
    - PTHREAD_PRIO_INHERIT
    - PTHREAD_PRIO_PROTECT
- int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t* attr, int* prioceiling);
- int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr, int prioceiling);

THRijswijk