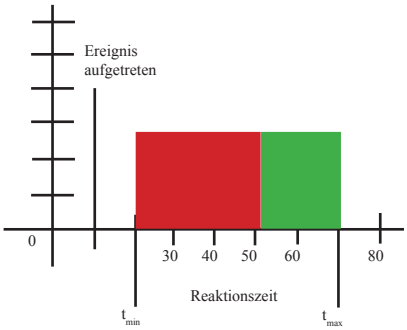
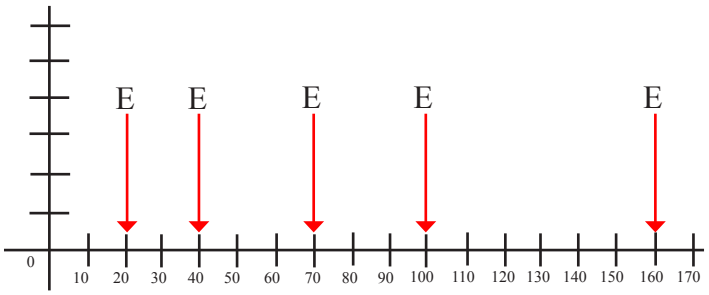


Reaktionsbereich : $t_{min} \leq t_{Reaktion} \leq t_{max}$

Anforderungsfunktion:



$t_{p\_min} = 2T$

$t_{p\_max} = 6T$

$t_{\bar{p}} = \frac{2T + 6T}{2} = 4T$

Gesamtauslastung $u = \sum_{i=1}^{k} \frac{t_{ei}}{t_{pi}} \leq 1$

Prioritätenvergabe:
Tasks mit kurzer Ausführungszeit haben in der Tegel auch kurze Prozesszeiten und bekommen hohe Priorität. Tasks mit langen Asführungszeiten haben in der Regel auch lange Prozesszeiten und bekommen niederige Priorität.

Glossar:

Determiniertheit:
nter zeitlichem Determinisumus ist die Berechenbarkeit des Zeitverhaltens des Rechensystems zu verstehen.

Worst Case:
Systemzustand, bei dem bei allen Jobs das Maximum an Reaktionszeit auftritt.
T_{pmin}
T_{pmax}
Anforderungen liegen zum Zeitpunkt 0 vor

Ressourcen:
aktive: Prozessor (P)
passive: Speicher, Sequenznummern, Semaphoren, ... (R)

Temporale Paramerter:

Funktionale Parameter:
-Unterbrechbarkeit (Preemptivität):
Die Ausführung eines Jobs auf einem Prozessor kann unterbrochen werden.
-Kontextwechsel (context switch):
-Priorität

Auslösezeitpunkt r
Deadline d
Ausführungszeit e

$\varnothing_i$ = **Phase**

p_i = **Periode** = $\frac{1}{f}$

e_i = **Ausführungszeit**

D_i = **relativeDeadline**

d = absolute Deadline

H = Hyperperiode = kgV aller p

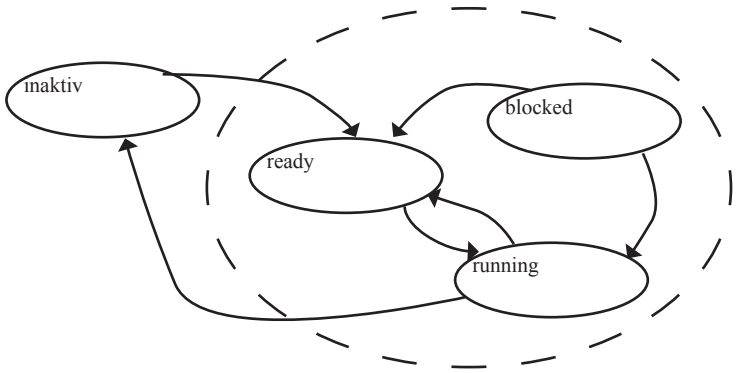
$(\varnothing_i, p_i, e_i, D_i)$, (p_i, e_i, D), $(p_i, e_i)p = D$,

$(r,d]$, $e(r,d]$ **bei Ereignisgesteuerten**

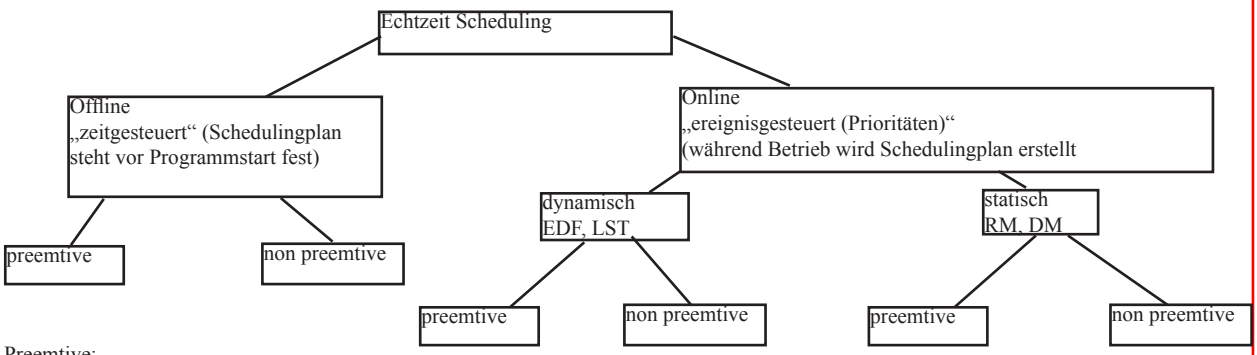
$(t_k, T(t_k)) = (t_{anfang\ Task}, T_{Dauer\ des\ Tasks}(t_k))$

$\lceil\ \rceil$ = **obere Gaussklammer 1.3->2**

$\lfloor\ \rfloor$ = **untere Gaussklammer 1.3->1**



<u>running:</u>
Dem Job ist ein Prozessor zugeteilt. Dieser Zustand wird verlassen durch:
-Der Job wird bis zum Ende ausgeführt -> beendet sich selbst
-Die Bedingungen für den Ablauf sind nicht mehr erfüllt (Ressourcen nicht mehr verfügbar) -> blocked
-Dem Job wird der Prozessor entzogen -Y ready
<u>blocked:</u>
Job wartete auf eine definierte Zeitspanne ider einer Ressource -> ready
<u>ready:</u>
Job ist lauffähig



Preemtive / non Preemtive:
Bei preemtiven Algo kannd der Job jederzeit unterbrochen werden, um den Prozessor einem anderen Job/Task zuzuordnen.
Beim non preemtive Algo wird der gestartete Job bis zu seiner Vollendung ausgeführt. Scheduling bei beendigung des Jobs /Tasks.

Statisch:
Schedulingentscheidungen basieren auf fixen Parametern der Jbs, die vor dem Start des Rechensystems bekannt sind.

Dynamisch:
Schedulingentscheidungen bsieren auf dynamischen Parametern, welche sich während der Laufzeit des Rechensystemes änder können.

Offline:
Die Entscheidung über die Reihefolge des Ablaufs ist vor dem Systemstart berechnet. Schedulingplan wird vorab berechnet.

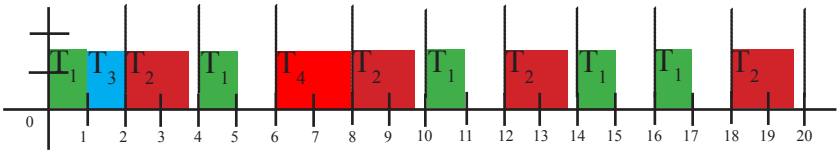
Online:
Das Scheduling erfolgt jeweils zur Laufzeit des Rechensysteme, immer dann wenn ein Job zur Ausführung bereitsteht.

Zeitgesteuertes Scheduling:

Konstante Framelänge (zyklisches Scheduling):
Konstante Periode, Framelänge f.
Beispiel:
{T1=(4;1);T2=(5,1,8);T3=(20,1);T4=(20,2)}
1.Bedingung f≥2
2.Bedingung f = 2,4,5,10,20
3.Bedingung alle Möglichen f müssen für jede Task überprüft werden, sobald bei einer Task unpassend -> f kommt nicht in Frage
Resultat : f = 2

Daraus ergibt sich folgende Schedulingtabelle:

Frame	Taskset
1	T1,T2
2	T2
3	T1
4	T4
5	T2
6	T1
7	T2
8	T1
9	T1
10	T2



Falls kein f gefunden wird, muss der Task mit der längsten Periode in 3 Teiltasks zerlegt werden Bsp.:
T={(4;1),(5;2;7),(20;5)} 1.Bedingung f ≥ 5 3.Bedingung f ≤ 4
Lösung : T={(4;1),(5;2;7),(20;1),(20;3),(20;1)}

Slack Stealing:
Um die Reaktionszeit zu verbessern, besteht die Möglichkeit, die sporadischen Jobs vor periodischen Tasks auszuführen. Periosiche Task wird nach hinten verschoben. Die komplette Schedulingtabelle kann wieder offline erechnet werden.

Vorteile:
einfaches Konzept: Es ist nur eine Schedulingtabelle nötig.
Zusätzliche Mechanismen für die kontrolle der nebenläufigkeit und synchronisationsmittel sind nicht nötig.
Framegrenzen und Zeitbedingungen können überprüft / erzwungen werden.
Einfach zu validieren, testen und zertifizieren.
Daher bevorzugt in Systemen mit hoher Sicherheitsaforderung

Nachteile:
Die Realeasetime der Jobs muss konstant sein
somit sind die sporadischen Jobs nur sehr schwer zu handhaben
Bei jeder Änderung muss die Schedulingtabelle neu berechnet werden
Anzahl der Frames einer Hyperperiode kann sehr lang werden.

Das Aufsplitten von Task mit einer langen Ausführungszeit in Subtasks ist fehleranfällig.

Achtung:
braucht nicht unbedingt Hyperperiode
Task evt. splitten

Bestimmen der Framelänge f:

Hyperperiode $H = kgv(\text{aller } p_i)$

1.Bedingung

$f \geq \max_{1 \leq i \leq n}(e_i)$

2.Bedingung:

$F = \frac{H}{f}$ **F=Anzahl der Frames f in Hyperperiode H** $F \in \mathbb{N}$

$f = \frac{H}{\mathbb{N}}$, **wobei das Resultat ganzzahlig ein muss**

3.Bedingung $f \leq \min(D_i)$**:**

$2 * f - GGT(p_i, f) \leq D_i$

Beispiel:

$T1 = (4,1), \quad T2 = (5,1,8), \quad T3 = (20,1), \quad T4 = (20,2)$

1.Bedingung: $f \geq 2$

2.Bedingung: $f = 2,4,5,10,20$

3.Bedingung alle möglichen f müssen mit allen Tasks geprüft werden:

$f = 2$:

$T1: 4 - GGT(4,2) = 2 \leq 4$ ->o.k

$T2: 4 - GGT(5,2) = 3 \leq 5$ ->o.k

$T3: 4 - GGT(20,2) = 2 \leq 20$ ->o.k

$T4: 4 - GGT(20,2) = 2 \leq 20$ ->o.k

$f = 4$:

$T1: 8 - GGT(4,4) = 4 \leq 4$ ->o.k

$T2: 8 - GGT(5,4) = 7 \leq 5$ ->nicht o.k

$f = 5$:

$T1: 10 - GGT(4,5) = 9 \leq 4$ ->nicht o.k

$f = 10$:

$T1: 20 - GGT(4,10) = 18 \leq 4$ ->nicht o.k

$f = 20$:

$T1: 40 - GGT(4,20) = 3 \square \leq 4$ ->nicht o.k

□omit ist die Framegrösse 2

Prioritätengesteuertes Scheduling (Online, Ereignisgesteuert):

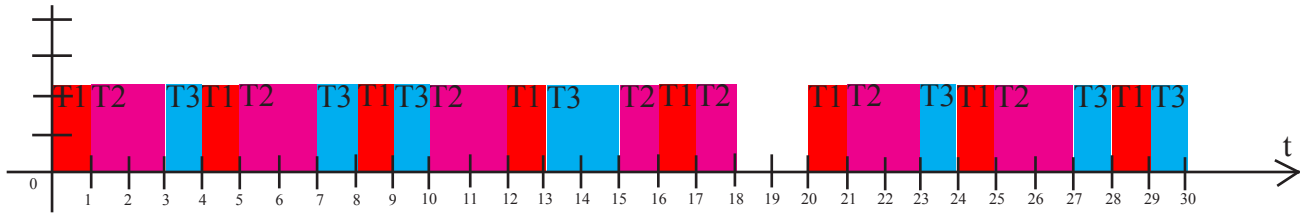
RM (Rate Monotonic) / Deadline Monotonic Algorythmus [Prioritätskriterium p] -> feste Priorität:

Die Tasks werden ihrer Priorität nacheineander ausgeführt. Priorität nach der Periode : je kürzer die Periode, je höher die Priorität.

Beim Schedulinglan : Task nach ihrer Periode der Reihe nach aufzeichnen. Tasks, die Deadline verpassen -> neuer Task beginnen

Bsp:

T={T1(4;1), T2(5;2), T3(20;5)} Prioritäten : T1,T2,T3



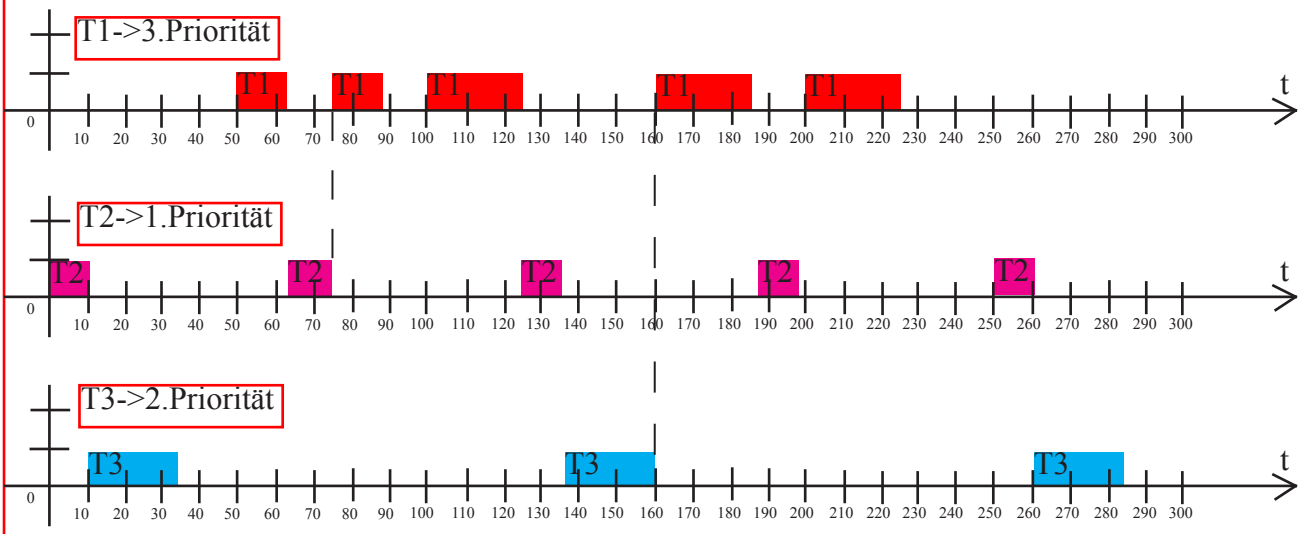
DM (Deadline Monotonic) [Prioritätskriterium D] -> feste Priorität:

Umso kürzer die relative Deadline, umso höher die Priorität.

Bsp:

Notation($Ø_i$,p_i,e_i,D_i)

T={T1(50;50;25;100), T2(0;62.5;10;20), T3(0;125;25;50)} Prioritäten:T2,T3,T1



wichtig:

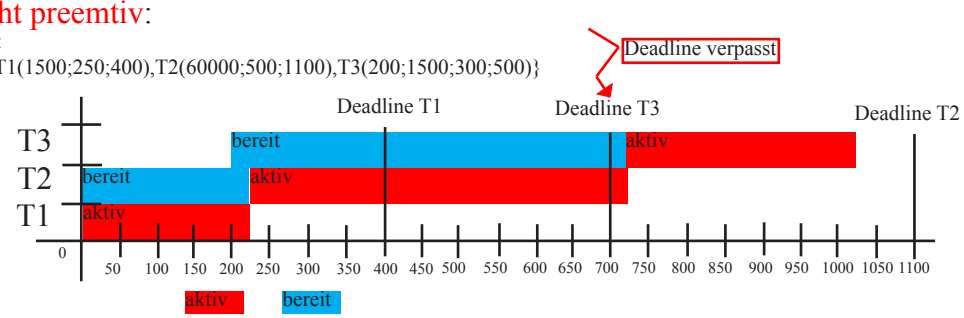
Sind Deadline und Perionde der Tasks nicht proportional, erreicht der DM-Algorythmus eher einen korrekten Schedulingplan als der RM-Algorythmus. Der RM-Algorythmus schlägt immer fehl, wenn auch der DM-Algorythmus fehlschlägt.

Der RM und DM Algorytmus arbeitet nur dann optimal, wenn die Perioden der Tasks ganzzahlige Vielfache zuwinander sind.

FIFO / FCFS (First In First Out / First Come First Serve)[Prioritätskriterium r] -> dynamische Priorität, nicht preemtiv:

Bsp.:

T={T1(1500;250;400),T2(60000;500;1100),T3(200;1500;300;500)}



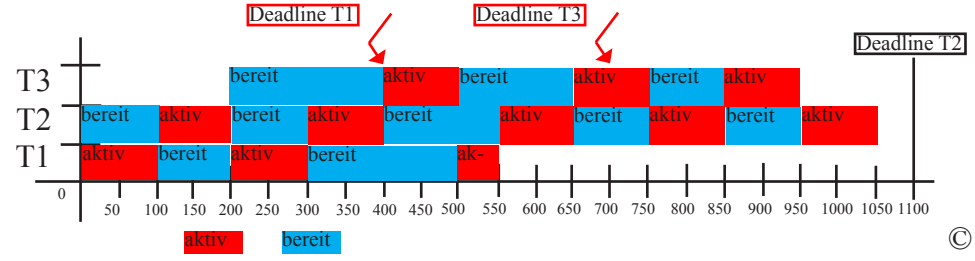
Round Robin [Prioritätskriterium (statisch, Quantum)] -> dynamische Priorität, nicht preemtiv:

ready Warteschlange: z bestimmten Zeitpunkten (Time Slice) wählt der Scheduler den Job am Kopf der Schlange zur bearbeitung aus.

Wenn Job in der Timeslice nicht fertig wird, kommt er ans Ende der Schlange. Ein neuer Job kommt ans Ende der Schlange.

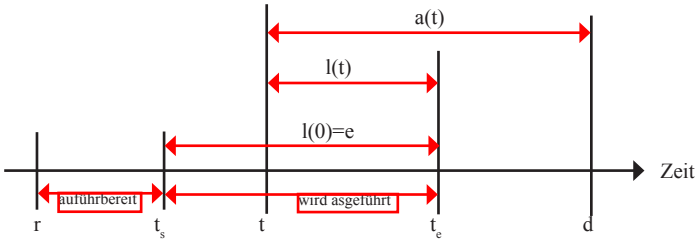
Schedulingpan : Zeit & Warteschlange

T={T1(1500;250;400),T2(60000;500;1100),T3(200;1500;300;500)}



EDF (Earliest Deadline First) [Prioritätskriterium d] -> dynamische Priorität:

Deadline des Jobs zum Zeitpunkt t: umso früher die Deadline, desto höher die Priorität.
Bsp:
T={T1(2;0.9),T2(5;2.3)}
Zum Zeitpunkt 0 ist die absolute Deadline von $T_{1,1}$ 2 und $T_{2,1}$ 5. Also erhält T1 eine höhere Priorität.
Zum Zeitpunkt 2 wird T1,2 rechenbereit mit D=4 also kleiner als $T_{2,1}$. $T_{1,2}$ erhält eine höhere Priorität und unterbricht $T_{2,1}$.
Zum Zeitpunkt 2.9 ist $T_{1,2}$ beendet und $T_{2,1}$ wird ausgeführt
Zum Zeitpunkt 4 wird $T_{1,3}$ rechenbereit mit D=6. $T_{1,3}$ erhält aufgrund der gegenüber $T_{2,1}$ späteren Deadline eine niederigere Priorität als der gerade ausgeführte Job. $T_{2,1}$ wird nicht unterbrochen, erst am Ende von $T_{2,1}$ wird $T_{1,3}$ ausgeführt.



r=Realease Time
ts=Tatächliche Startzeit
te=Tatsächliche Endzeit
d=Deadline
t=Beobachtungszeitpunkt
a(t)=Antwortzeit
l(t)=restliche Ausführungszeit (noch verbleibende Laufzeit)
e=Ausführungszeit

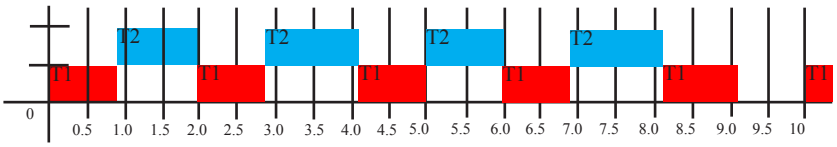
Zeitpunkt t	EDF-Queue	Deadlines
0	T1,T2	2,5
0.9	T2	5
2	T1,T2	4,5
2.9	T2	5
4	T2,T1	5,6(!!)
4.1	T1	6
5	T2	10
6	T1,T2	8,10
6.9	T2	10
8	T2,T1	10,10(FIFO)
8.2	T1	10
9.1	Idle	Idle

Es sei eine Menge von Jobs gegeben, die zur Bearbeitung anstehen, wobei alle sofort bearbeitbar seinen. Die Indizierung ist so gewählt, dass gilt:

$$a_i(t) \le a_{i+1}(t) \qquad 1 \le i \le n-1$$

Ein korrekter Schedulingplan kann genau dann gefunden werden, wenn gilt:

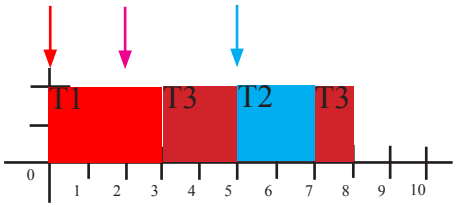
$$a_k(t) \ge \sum_{i=1}^{k} l_i(k) \qquad 1 \le k \le n$$



Damit EDF optimal arbeitet, müssen alle Jobs unterbrechbar sein, Ausserdem arbeitet EDF nur auf einem Einprozessorsystem optimal.

LST (Least Slack Time) [Prioritätskriterium d,e] (Theroetisches Modell -> fast nie in der Praxis) -> dynamische Priorität:

Zu jeder Zeit t berechnet sich die Slacktime (laxity):d-t-(restliches e). Mit restlichem e ist die noch verbleibende Executiontime gemeint. Umso kleiner die Slacktime, umso grösser die Priorität.
Bsp.:
T1=3(0,6]
T2=2(5,7]
T3=3(2,8]
T1 hat zum Zeitunkt t=0 eine Slack Time von 3. Nun wird zum Zeitpunkt t=2 T3 aktiv. Der Scheduler wird aufgerufen, der die Prioritäten neu bestimmt.
T3 hat bei t=2 ebenfalls eine Slacktime von 3 -> Eintrag in Warteschlange Slacktime=3. T1 wird weitergeführt.
Nach Beendigung von T1 wird T3 ausgeführt.
Zum Zeitpunkt t=5 wird T2 aktiv. Die Slacktime ergibt sich zu 0, somit unterbricht T2 T3 aufgrund der höheren Priorität.
Bei t=7 hat T3 ein Slacktimer von 0.



Echtzeitnachweis bei statischer Priorität (RM oder DM):

1. Test (falls dieser O.K ist die Echzeit nachgewiesen, falls nicht -> 2.Test):

$$\mu = \sum_{k=1}^{i} \frac{e_k}{\min(D_k, p_k)} \leq i * \left(2^{\frac{1}{i}} - 1\right)$$

2.Test TDA (Time Demand Analysis):

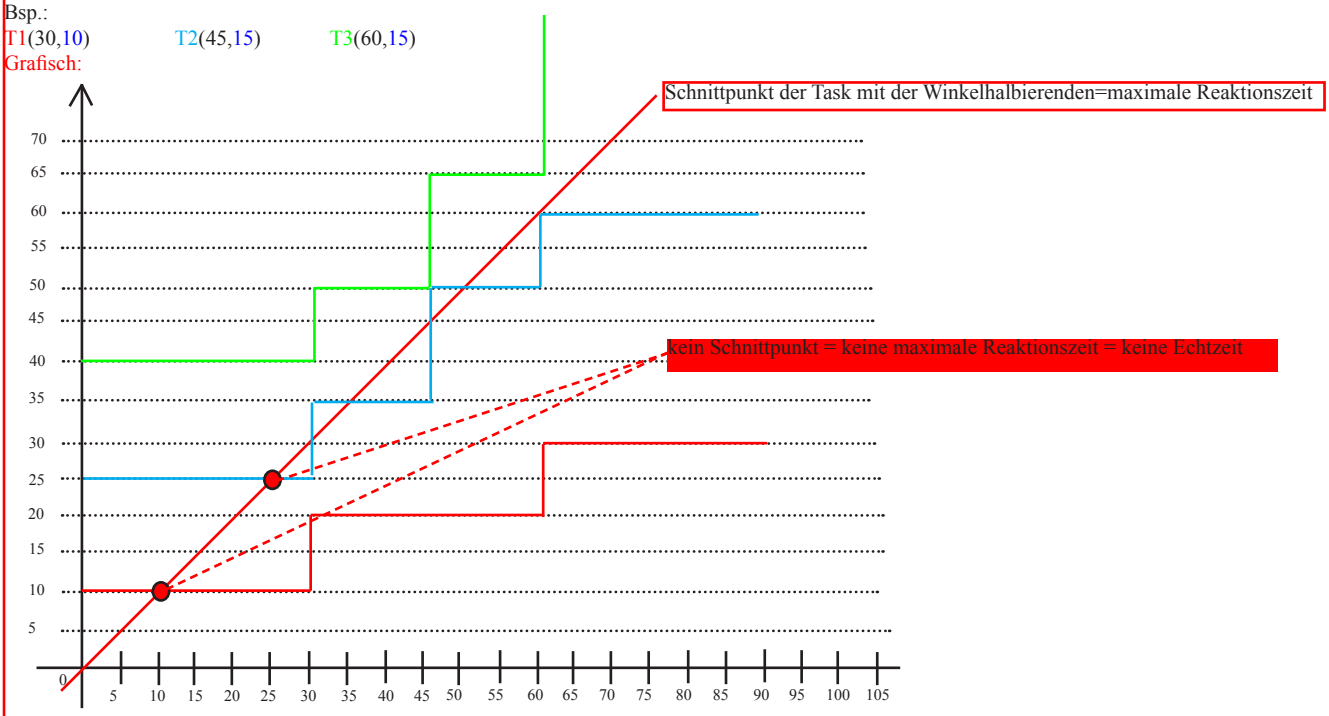
$$t^{(l+1)} = \sum_{k=1}^{i} \left\lceil \frac{t^{(l)}}{p_i} \right\rceil * e_k$$

$e_i \leq t \leq \min(D_i, p_i)$ = **Echtzeit**

$t > \min(D_i, p_i)$ = **keine Echtzeit**

$t^{l+1} = t^l$ = **Echtzeit**

Bsp.:
T1(30,10) T2(45,15) T3(60,15)
Grafisch:



Mathematisch:

t = maximale Reaktionszeit = maximale Ausführungszeit – minimale Reaktionszeit

mit e starten als $t^0 = e$

Task1=e

Task2=Rechnung(Task1 + Task2)

$$t^{(l+1)} = \sum_{k=1}^{i} \left\lceil \frac{t^{(l)}}{p_i} \right\rceil * e_k$$ → **obere Gaussklammer**

T1:

$t^0 = 10$

$t^2 = 10$

T2:

$t^0 = 15$

$$t^1 = 15 + \left\lceil \frac{15}{30} \right\rceil * 10 = 25$$

$$t^3 = 15 + \left\lceil \frac{25}{30} \right\rceil * 10 = 25$$ $t^l = t^{l+1}$=Echtzeit

T3:

$t^0 = 15$ = **Ausführungszeit**

$$t^1 = 15 + \left\lceil \frac{15}{30} \right\rceil * 10 + \left\lceil \frac{15}{\square 5} \right\rceil * 15 = \square 0$$

$$t^2 = 15 + \left\lceil \frac{\square 0}{30} \right\rceil * 10 + \left\lceil \frac{\square 0}{\square 5} \right\rceil * 15 = 50$$

$$t^3 = 15 + \left\lceil \frac{50}{30} \right\rceil * 10 + \left\lceil \frac{50}{\square 5} \right\rceil * 15 = \square 5$$

$$t^{\square} = 15 + \left\lceil \frac{\square 5}{30} \right\rceil * 10 + \left\lceil \frac{\square 5}{\square 5} \right\rceil * 15 = \square 5$$

$$t^5 = 15 + \left\lceil \frac{\square 5}{30} \right\rceil * 10 + \left\lceil \frac{\square 5}{\square 5} \right\rceil * 15 = \square 5$$ $t \square \min(\square_i \square p_i) \rightarrow$ **keine Echtzeit**

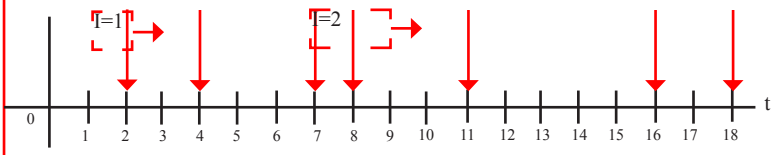
Echtzeitnachweis bei dynamischer Priorität (EDF) Ereignisströme:

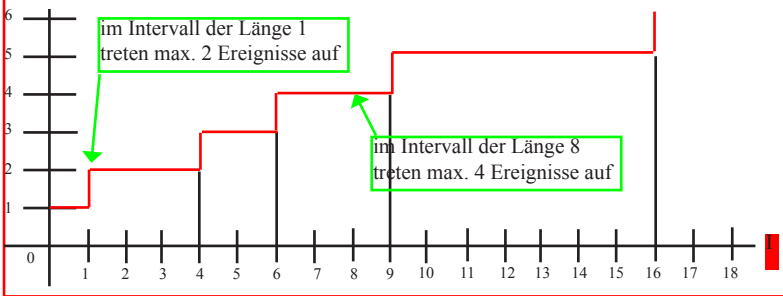
1. Test (falls dieser O.K ist die Echzeit nachgewiesen, falls nicht -> 2.Test):

$$\sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} \le 1$$

2.Ereignisstrommodell:

1.Ereignisstrommodell

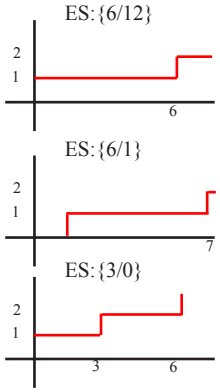


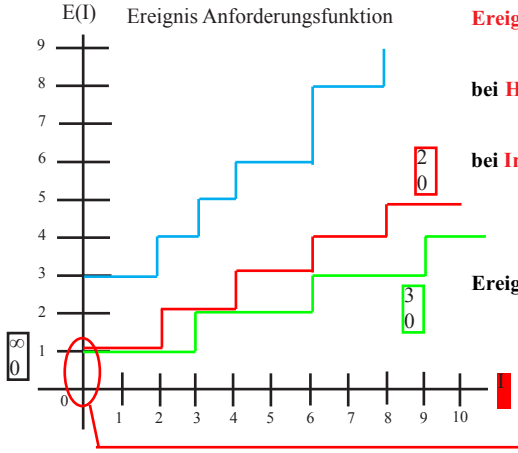


Inhomogen → homogen

Inhomogen: $\left\{\begin{pmatrix}6\\0\end{pmatrix},\begin{pmatrix}6\\1\end{pmatrix},\begin{pmatrix}3\\0\end{pmatrix}\right\}$ $H = kgV(6,6,3) = 6$

Homogen: $\left\{2*\begin{pmatrix}6\\0\end{pmatrix},\begin{pmatrix}6\\1\end{pmatrix},\begin{pmatrix}6\\3\end{pmatrix}\right\}$





Ereignisstromtupel $ES: \left\{\begin{pmatrix}\text{Hyperperiode } p_i\\ \text{Intervall } a_i\end{pmatrix}\right\}$

bei Homogenen Ereignisstrom, alle Perioden p_i gleich. Bsp.: $\left\{\begin{pmatrix}16\\0\end{pmatrix},\begin{pmatrix}16\\1\end{pmatrix},\begin{pmatrix}16\\4\end{pmatrix},\begin{pmatrix}16\\6\end{pmatrix},\begin{pmatrix}16\\9\end{pmatrix}\right\}$ (Periode steigt um 1)

bei Inhomogenem Ereignisstrom, Perioden $p_i$unterschiedlich. Bsp.: $\left\{\begin{pmatrix}\\0\end{pmatrix},\begin{pmatrix}\\0\end{pmatrix},\begin{pmatrix}\\0\end{pmatrix}\right\}$ (□o steigen um e beginnt bei)

$$ES: \left\{\begin{pmatrix}16\\0\end{pmatrix},\begin{pmatrix}16\\1\end{pmatrix},\begin{pmatrix}16\\4\end{pmatrix},\begin{pmatrix}16\\6\end{pmatrix},\begin{pmatrix}16\\9\end{pmatrix}\right\}$$

Ereignisse : 1 □ □ 4 □

Inhomogen ES $= \left\{\begin{pmatrix}\infty\\0\end{pmatrix},\begin{pmatrix}2\\0\end{pmatrix},\begin{pmatrix}3\\0\end{pmatrix}\right\}$

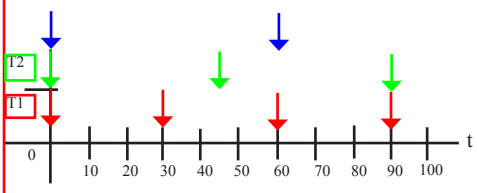
Hyperperiode $= GKV(2,3) = 6$

Homogen ES $= \left\{\begin{pmatrix}\infty\\0\end{pmatrix},2*\begin{pmatrix}6\\0\end{pmatrix},\begin{pmatrix}6\\2\end{pmatrix},\begin{pmatrix}6\\3\end{pmatrix},\begin{pmatrix}6\\□\end{pmatrix}\right\}$

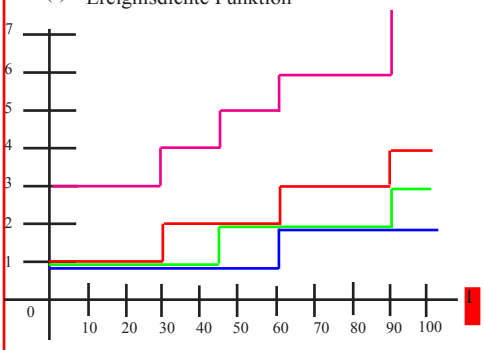
Bsp:

Anforderung	ei	pi	Di
1	10	30	30
2	15	45	45
3	15	60	60

Ereignis Anforderungsfunktion



E(I) Ereignisdichte Funktion



T1 : p = 30

T2 : p = 45

Hyperperiode = 90

$$ES = \left\{\begin{pmatrix}30\\0\end{pmatrix},\begin{pmatrix}45\\0\end{pmatrix},\begin{pmatrix}60\\0\end{pmatrix}\right\}$$

p_i = Periode

a_i = Phase

$$E(I) = \left\lfloor \frac{I + p_i - a_i}{p_i} \right\rfloor$$

$$W(I) = \sum E_i * (I - D_i) * e_i$$

$$= \sum \left\lfloor \frac{I - D_i + p_i - a_i}{p_i} \right\rfloor * e_i$$

$0 \le I \le$ Hyperperiode

$W(I) \le I$

bei mehreren Tasks $W(I)_{ges} \le I$

Auslastung:

$W(0) = 0$

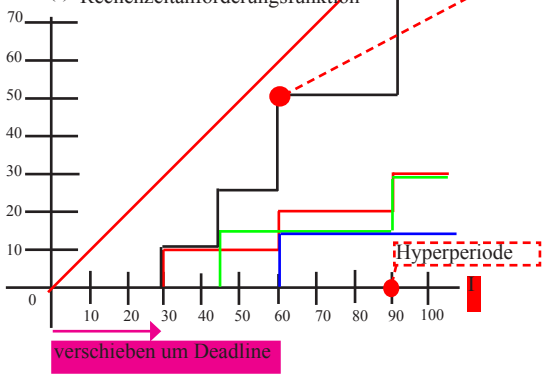
$W(30) = 10$

$W(45) = 25$

...

$W(60) = 50 \rightarrow$ Auslastung $= \frac{50}{60} = 0.8333$

W(I) Rechenzeitanforderungsfunktion



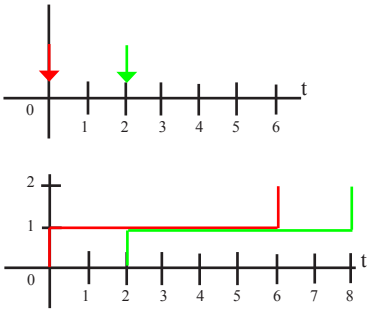
Kritischer Punkt bei 60 (Phase=0, Deadline=Periode):

$$W_{ges} = \left\lfloor \frac{I}{30} \right\rfloor * 10 + \left\lfloor \frac{I}{30} \right\rfloor * 15 + \left\lfloor \frac{I}{30} \right\rfloor * 15$$

$$W_{ges}(60) = \left\lfloor \frac{60}{30} \right\rfloor * 10 + \left\lfloor \frac{60}{30} \right\rfloor * 15 + \left\lfloor \frac{60}{30} \right\rfloor * 15 = 50$$

$50 \le 60 \rightarrow$ o.k

Abhängige Tasks:



$T1(6,2,3) \quad (p_i, e_i, D)$

$T2(2,6,3,4) \quad (\varnothing_i, p_i, e_i, D_i)$

$$W(I) = \sum \left\lceil \frac{I - D_i + p_i - a_i}{p_i} \right\rceil * e_i$$

$$W(I) = \left\lceil \frac{I-3+6-0}{6} \right\rceil * 2 + \left\lceil \frac{I-4+6-2}{6} \right\rceil * 3$$

$$W(I) = \left\lceil \frac{I+3}{6} \right\rceil * 2 + \left\lceil \frac{I}{6} \right\rceil * 3$$

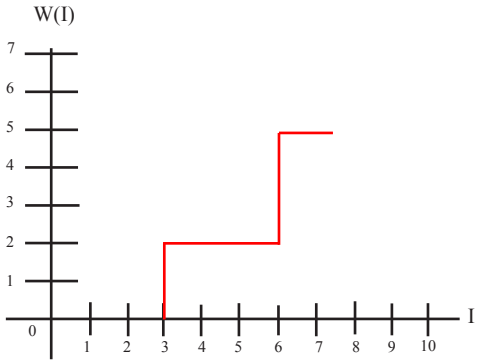
kritische Punkte:

$W(3) = 2$

$W(6) = 5$

$W(9) = 7$

$W(I) \leq I \rightarrow$ **o.k**



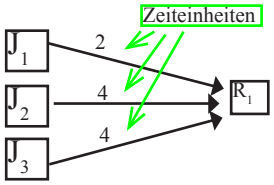
Ressourcen:

Darstellung:

L(R,3) Lock Ressource 3

U(R,3) Unlock Ressource 3

$J_1[R_3, 10[R_2, 5]]$ Job 1 greift auf Ressource 3 10 Zeiteinheiten zu...



Problem : Prioritätsinversion (hochprioritärer Job wartet auf niederprioren Job)

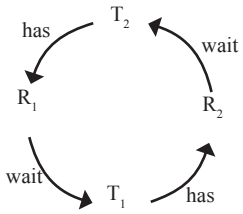
Deadlock, wenn :

1.Ressourcen sind nur exclusive nutzbar (Mutual Exclusion)

2.keine Preemtivität (Task kann nicht einfach unterbrochen werden! -> no preemtion)

3.Tasks halten die Ressource auch dann, wenn sie auf die Zuweisung weiterer Ressourcen warten. (hold-and-wait)

4.Es gibt eine zyklische Kette von Tasks, von denen jede mind. eine Ressource besitzt, die die nächste Task benötigt (circular wait)



e(r;D]

Bankers Algorythmus (Prüfung, ob es zum Deadlock kommt):

R_{ges} ->wieviel Ressourcen insgesamt

A	B	C
10	5	7

M_{max} Wieviel Ressourcen werden von den Tasks maximal gebraucht

	A	B	C
T_0	7	5	3
T_1	3	2	2
T_2	9	0	2
T_3	2	2	2
T_4	4	3	3

U derzeitige Belegung zum Zeitpunkt T

	A	B	C
T_0	0	1	0
T_1	2	0	0
T_2	3	0	2
T_3	2	1	1
T_4	0	0	2
total	7	2	5

$RR=R_{ges}-U_{total}$

A	B	C
3	3	2

$M=M_{max}-U$

	A	B	C	
T_0	7	4	3	kann mit RR zu Ende laufen
T_1	1	2	2	kann mit RR zu Ende laufen
T_2	6	0	0	kann mit RR zu Ende laufen
T_3	0	1	1	kann mit RR zu Ende laufen
T_4	4	3	1	kann mit RR zu Ende laufen

neues RR = RR + T_1

A	B	C
5	3	2

neues RR = RR + T_4

A	B	C
5	3	4

neues RR = RR + T_3

A	B	C
7	4	5

neues RR = RR + T_0

A	B	C
7	5	5

neues RR = RR + T_2

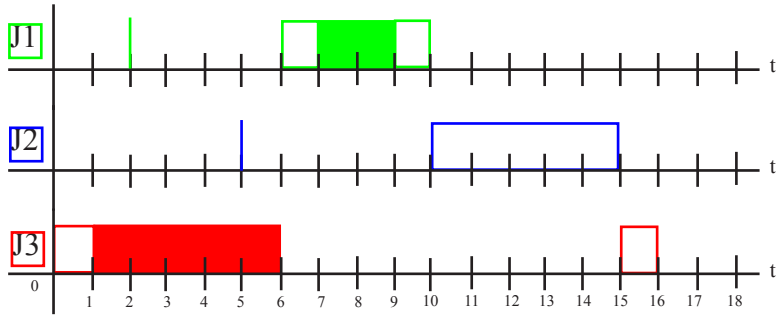
A	B	C
10	5	7

Bei mindestens 2 Tasks, die nicht fertig werden -> Deadlockgefahr
Bei 1 Task, der nicht fertig wird -> Rechenfehler
Falls alle Tasks fertig werden -> keine Deadlockgefahr

Protokolle zur Behandlung der Prioritätsinversion:

NPCS Protokoll (Non Preemtive Critical Section) Nichtunterbrechbare kritische Bereiche:

verhindert Deadlock!
Interrptsperre nur im kritischen Bereich.



J1 muss nun auf J3 warten, wird also erst ausgeführt, nachdem J3 den kritischen Bereich verlassen hat,
der ja nun nicht unterbrechbar ausgeführt ist. Den Fall der unbegrenzten Blockierzeit kann es bei dieser Lösung nicht geben. Ebenfalls kann bei NPCS kein Deadlock mehr auftreten.

Wann Verwendung:
Für statische sowie dynamische Prioritäten.
Wenn kritische Bereiche sehr kurz, und es zwischen den Tasks/Jobs im System viele Ressourcenkonflikte gibt.

Vorteil:
Einfach zu implementieren. Es werden keine Informationen über die Ressourcen Anforderungen der Tasks/Jobs vorab benötigt.

Nachteil:
Jeder hochpriorer Job wird bis zum Ende des kritischen Bereichs blockiert, auch wenn dieser hochpriore Job gar keinen Zugriff auf die Ressource braucht.

Blockierzeit b_i :

$$b_i(rc) = \max_{i+1 \le k \le n}(c_k) = \max_{i+1 \le k \le n}(\text{Kritischer Bereich})$$

PIP (Priority Inheritance Protocol):

verhinder Deadlock nicht!
Verhindert, dass eine Job/Task lange auf eine Ressource warten muss.
Zwei Arten von Blockieren: direktes Blockieren oder Blockiern durch die Prioritätenvererbung
Verhinder unkontrollierte Blockierzeit, jedoch die gesamte Blockierzeit nicht minimiert.
Scheduling Regel:
-Entsprechend der Prioritäten durch den Scheduling Algorythmus werden die Jobs ausgeführt.
Die tatsächliche Priorität πi der jobs ist also bei Start des Jobs (also zur Release Zeit) gleich der Priorität, die der Scheduling Algorythmus berechnet hat.
-Jeder Job wird unterbrechbar entsprechend seiner Priorität geschedult.
Ressourcen Zuteilungs Regel:Wenn ein Job zur Zeit t eine Ressource anfordert,
-und R frei ist, erhält Job die Tessource und behält diese solange, bis Job sie wieder freigibt.
-und R nicht frei st, wird die Anforderung abgelehnt und Job blockiert.
Prioritäten Vererbungs Regel:
Wenn der Job J, der die Ressource angeforder hat blockiert wird, so erbt der Job Jl, der die Ressource besitzt vorübergehend die Priorität von J.
Jl wir mit dieser Priorität ausgeführt, bis er wieder die Ressource abgibt. Der Job Jl erhält dann wieder die Priorität, die er vor dem Zugriff auf die Ressource hatte.
Bsp.:

Job	ri	ei	πi
J1	7	3	1
J2	5	3	2
J3	4	2	3
J4	2	6	4
J5	0	6	5

Regeln im PIP:

1.Scheduling Regel

π_i **wird zur Releasezeit festgelegt.** $(\pi_i = \textbf{Priorität})$

Jeder Job wird unterbrechbar entsprechend seiner Priorität geschedult.

Prioritätenbasiertes Scheduling

2.Ressurcen Zuteilung:

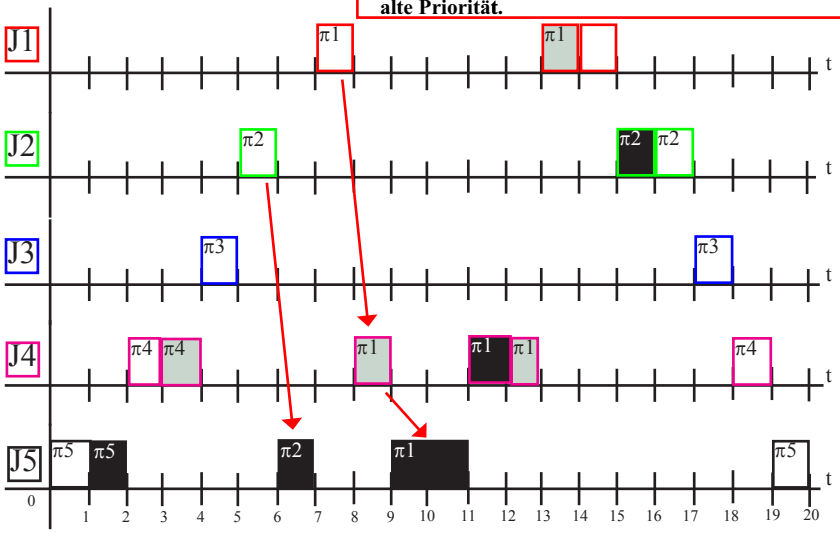
$\mathbf{R_{frei} : R \rightarrow J}$ **(J bekommt Ressource und behält sie solange bis J sie wieder freigibt)**

$\mathbf{R_{nicht\ frei}}$ **: J blockieren**

3.Prioritäten vererben:

Wenn J, der die Ressource angefordert hat blockiert wird, erbt J_L, der die Ressource besitz vorübergehend die Priorität von J.

J_L wird mit dieser Priorität ausgeführt, bis er die Ressource abgibt. Danch hat er wieder seine alte Priorität.



Zeit	Was
0	J5 ready und mit Priorität 5 ausgeführt. bei t=1 belegt J5 Ressource schwarz.
2	J4 wird rechenbereit. Da höhere Priorität wird er sofort ausgeführt. Unterbricht J5
3	J4 belegt die Ressource grau und wird weiter ausgeführt.
4	J3 wird rechenbereit und unterbricht J4
5	J2 realeased und unterbricht J3.
6	J2 führt einen L (schwarz) aus, versucht also auf die Ressource schwarz zuzugreifen. Der Zugriff kann nicht ausgeführt werden, da ja die Ressource bereits von J5 benutzt wird. J2 wird blockiert. Entsprechend der 3. Regel des PIP „erbt“ J5 die Priorität 2 von J2. Somit hat J5 jetzt die höchste Priorität alle rechenbereiter Jobs un wird ausgeführt.
7	J1 wird realeased . J1 hat die höchste Priorität und unterbricht somit J5
8	J1 versucht auf die Ressource grau zuzugreifen. Da dies nicht gelingt, wird J1 blockiert. J4 hat die Ressource, erbt somit die Priorität von J1 und hat nun die höchste Priorität aller rechenbereiten Jobs und wird ausgeführt.
9	J4 will auf die Ressource schwarz zugreifen, was fehlschlägt, wodurch J4 blockiert wird. Zu diesem Zeitunkt hat J4 die Priorität 1 die jetzt wiederum J5 erbt, wodurch J5 weiter mit Priorität 1 ausgeführt wird
11	J5 gibt die Ressource schwarz frei. J5 bekommt nun wieder die Priorität, die er vor dem Zugriff auf die Ressource hatte, also 5. Nun hat J4 die höchste Priorität (1) unter den rechenbereiten Jobs, da ja die Ressource schwarz wieder frei ist. J4 wird weiter ausgeführt und gibt erst die Ressource schwarz frei und dann die Ressource grau.
13	Ressource grau wird freigegeben. somit bekommt J4 seine frühere Priorität die er vor den Zugriff hatte, nähnlich 4 zurück.J1 wird rechenbereit gesetzt, hat die höchste Priorität und wird dadurch ausgeführt
15	J1 ist beendet. J2 hat die höchste Priorität und Tasks belegen die Ressoucen auch dann, wenn die auf die Zuweisung weiterer Ressorcen warten (Hold and wait). kann auf die Ressource schwarz zugreifen
17	Ab t=17 werden nach dem Ende von J2 alle Jobs entsprechend ihrer Priorität bis zum Ende ausgeführt.

PCP Protokoll (Priority Ceiling Protocol) :

Deadlock wird verhindert! Blockierzeit wird minimiert.

Ressourcen bekmmen eine Priorität.

2 Annahmen:

Die einmal den Jobs zugeteilten Priotität ist statisch (Schedulingverfahren mit statischer Priotität) .

Die von den Jobs benötigten Ressourcen sind vor dem Start bekannt.

Priority Ceiling:

Die Priority Ceiling einer jeden Ressource ist die Priotität des höchstprioren Jobs, der die Ressource benötigt.

Die aktuelle Priority Ceiling $\prod_s$ des Systems entspricht der höchsten Priorität der Priority Ceiling der Ressourcen, die gerade benutzt werden.

Wird keine Ressource benutzt wird $\prod_s = \Omega$ gesetzt, wobei Ω eine Priorität ist, die niederiger ist als die realen Prioritäten.

Regeln:

Berechnung der aktuellen Ceiling:

-Berechnung der aktuellen Ceiling des Systems. Sind alle Ressourcen frei, so gilt $\prod_s = \Omega$. π wird immer geupdatet, wenn eine Ressource belegt oder frei wird.

Scheduling Regel:

die tatsächliche Priorität π_s der Jobs ist beim Start des Jobs (also zur Release Zeit) gleich der Priorität, die der Scheduling Algorythmus berechnet hat. Die Jobs behalten diese Priorität. Ausnahme sind die Bedingungen der nächsten Regel.

Jeder Job wird unterbrechbar entsprechend seinerseiner Priorität.

Ressourcenzuteilungs Regel:

wenn ein Job zur Zeit t eine Ressource anfordert:

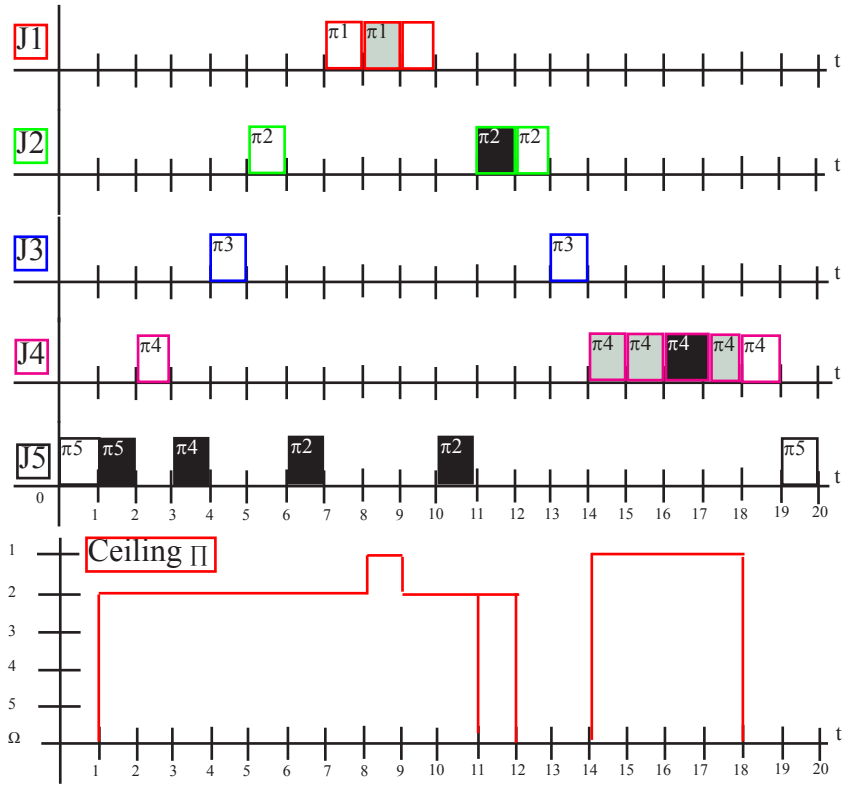
-und R nicht zur Verfügung steht, wird die Anforderung abgelehnt und J blockiert

-und R frei ist erhält J die Ressource, wenn J's Priorität $\pi(t)$ höher als $\prod_s(t)$ ist.

-Falls J's Priorität nicht höher als $\prod_s(t)$ ist erhält J die Ressource nur, wenn J der Job ist, durch dessen Ressource Zugriffe die derzeitige $\prod_s(t)$ ausgelöst wurde. Ansonsten wird die Ressource abgelehnt.

Prioritäten Vererbungsregel:

Wennn der Job J, der die Ressource angeforert hat blockiert wird, so erbt der Job, der zur Blockierung von J geführt hat (J_L) vorübergehend die Priorität $\pi(t)$ von J. Der Job J_L wird nun mit dieser Priorität ausgeführt, bis er jede Ressource abgibt, deren Priority Ceiling gleich oder höher der derzeitigen Priorität $\pi(t)$ von J_L ist. Der Job J_L erhält dann wieder die Priorität, die er vor dem Zugriff auf die Ressource hatte.



Job	ri	ei	πi
J1	7	3	1
J2	5	3	2
J3	4	2	3
J4	2	6	4
J5	0	6	5

Im Unterschied zu PIP zeigt dieses Verfahren, dass die höher prioren Jobs nun früher beendet werden können. Der Grund liegt in dem Blockieren des Jobs J4 zum Zeitpunkt t=3 aufgrund der Ressourcen Zuteilungsregel. Das Beispiel zeigt auch, dass es 3 Arten des Blockierens gibt:

-Direktes Blockieren

-Blockiern durch die Prioritätenvererbung

-Blockieren durch die Priority Ceiling

Zeit	Was
1-3	Wie bei PIP. Das Ceiling des Systems st zu Beginn Ω. J5 belegt schwarze Ressource. $\prod_s$=2,
3	J4 versucht , auf die graue Ressource zuzugreifen. Zugriff abgewehrt und J4 blockiert, da $\prod_s$=2 höher als π(J4)=4. Da J5 für die derzeitige Priority Ceiling verantwortlich ist, und für die Blockierung von J4 verantwortlich ist, erbt J5 die Priorität von J4 und wird mit der Priorität $\prod$=4 ausgeführt.
4	J3 unterbricht J5
5	J2 unterbricht J3
6	J2 versucht auf die Ressource schwarz zuzugreifen und wird direkt blockiert. J5 erbt nun die Priorität von J2 und wird mit der Priorität 2 ausgeführt.
7	J1 wird released
8	J1 versucht auf die Ressource grau zuzugreifen und da die Priorität von J1 höher ist als die Ceiling Priorität des Systems, kann J1 die Ressource belegen und wird weiter ausgeführt.
10	J1 ist beendet. Die derzeit rechenbereiten Jobs sind J3 und J5. J5 hat derzeit die Priorität 2 und wird somit als nächstes ausgeführt.
11	J5 gibt die Ressource schwarz frei. J5 hat dadurch wieder die Priorität 5. Die Ceiling des Systems ist nun wieder Ω. J2, welcher auf die Ressource schwarz wartet wird rechenbereit und entsprechend seiner Priorität als nächster ausgeführt
14	Nachdem J2 und J1 beendet sind, kann entsprechend der Priorität nun J4 ad die Ressource grau zugreifen, da die Priorität von J4 höher als das Ceiling des Systemes ist. J4 wird ausgeführt, das Ceiling des Systemes wird entsprechend dem Priority Ceiling von grau auf 1 gesetzt
16	Nun versucht J4 auf die Ressource schwarz zuzugreifen. Die Priorität von J4 ist niedriger als die Ceiling des Systems, jedoch ist J4 der Job der die Ressource belegt, die für das derzeitige Ceiling des Systemes verantwortlich ist. Somit kann J4 entsprechend der Regel die Ressource schwarz belegen und weiter ausgeführt werden.
19	j4 ist beendet und J5 wird bis zum Ende ausgeführt.

Regeln von PCP:

Annahmen:

Die einmal den Jobs zugeteilte Priorität ist statisch.

Die von den Jobs benötigten Ressourcen sind vor dem Start bekannt.

Die Priority Ceiling $\pi(R)$ jeder Ressource ist die Priorität des höchstprioren Jobs, der diese Ressource benötigt.

Die aktuelle Priority Ceiling Π_s des Systems entspricht der höchsten Priorität der Priority Ceiling $\pi(R)$ der Ressource, die gerade von Jobs benutzt werden.

Wird keine Ressource benutzt so wird $\Pi_s = \Omega$ gesetzt, wobei Ω eine Priorität ist, die niederiger als die realen Prioritäten ist.

1.Berechnung der akuellen Ceiling im System:

Sind alle Ressourcen frei, so gilt $\Pi_s = \Omega$. Π_s wird immer geupdatet, wenn eine Ressurce belegt, oder freigegeben wird.

2.Scheduling Regel:

Die tatsächliche Priorität π_i der Jobs ist beim Start der Jobs gleich der Priorität, die der Schedulingalgorythmus berechnet hat.

Die Jobs behlten diese Priorität mit ausnahme Regel 3.

Jeder Job unterbrechbar entsprechend seiner Priorität geschedult.

3.Ressourcen Zuteilung:

Wenn ein Job J eine Ressource R anfordert:

-und R nicht verfügbar ist, wird die Anforderung abgelehnt und J blockiert

-und R frei ist:

-erhält J die Ressource wenn J's Priorität π(t) höher ist als Π_s(t) $\pi_J > \hat{\pi}$.

-Falls nicht, erhält J die Ressoure nur, wenn J für die Π_s(t) verantwortlich ist.

-Falls nicht -> Anfrderung abgelehnt und J blockiert.

Prioritäten vererbung:

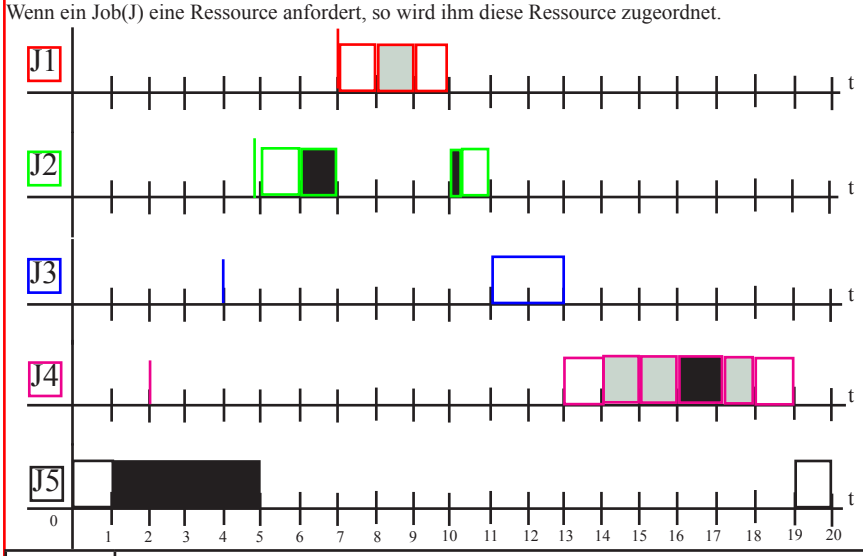
Wenn J, der die Ressource angefordert hat blockiert wird, erbt J_L, der die Ressource besitz vorübergehend die Priorität von J.

J_L wird mit dieser Prioriät ausgeführt, bis er die Ressource abgibt. Danch hat er wieder seine alte Priorität.

SPCP Protokoll (Stack-Based Priority Ceiling Protocol) :

wie PCP aber mit weniger Kontextwechsel.
Verhinder Deadlock. Gleiche Blockierzeiten wie bei PCP. Jedoch wird Job nur einmal blockiert und zwar bevor er ausgeführt wird.
So werden zusätzliche Kontextwechsel wie bei PCP oder PIP vermieden, auch wenn PCP und SPCP kürzere Blockierzeiten aufweist wie PIP, so wird dies nur erreicht, da die Ressourcenanfrderung aller Jobs/Tasks im System vorab bekannt sind!!!!!!!!
Regeln:
Die Priority Ceiling einer jeden Ressource ist die Priotität des höchstprioren Jobs, der die Ressource benötigt.
Die aktuelle Priority Ceiling $\prod_s$ des Systems entspricht der höchsten Priorität der Priority Ceiling der Ressourcen, die gerade benutzt werden.
Wird keine Ressource benutzt, wird $\prod_s=\Omega$ gesetzt, wobei Ω eine Priorität ist, die niederiger ist als die realen Prioritäten.

Die Regeln des SPCP sind:
1.Berechnung der aktuellen Ceiling des Systems:
Sind alle Ressourcen frei, so gilt $\prod_s=\Omega$. $\prod_s$ wird immer geupdated, wenn eine Ressource belegt oder freigegeben wird.
2. Scheduling Regel:
Nachdem der Job released ist, wird seine Ausführung solange blockiert, bis seine zugeordnete Priorität höher ist als die Ceiling des Systems.
Zu jeder Zeit werden Jobs, die nicht blockiert sind entsprechend ihrer zugeordneren Priorität nach einem prioritätsbasierten preemtiven Scheduling ausgeführt.
3.Ressourcenzuteilung:
Wenn ein Job(J) eine Ressource anfordert, so wird ihm diese Ressource zugeordnet.



Job	ri	ei	πi
J1	7	3	1
J2	4.8	3.2 (1.2 im kritischen Bereich)	2
J3	4	2	3
J4	2	6	4
J5	0	6	5

Zeit	Was
2	J4 wird realeased und wird sofort blockiert, da die Priorität nicht höher ist als die Ceiling des Systems (zu dieser Zeit ist $\prod_s=2$). J5 wir somit weiter ausgeführt
4	Aus gleichem Grund wird J3 blockiert.
4.8	Auch J2 wird aus diesem Grund blockiert.
5	Ressource schwarz wird von J5 freigegeben. Das Ceiling des Systems ist $\prod_s=\Omega$. Somit wird J2, der die höchste Priorität besitzt (2) ausgeführt. Wie erwartet kann er zum Zeitpunkt t=6 auf die Ressource schwarz zugreifen.
7	J1 wird released.Die Priorität von J1 ist häher als die Ceiling, somit wird J2 verdraängt und J1 ausgeführt. J1 liegt nun oben auf dem Runtimestack, und wird bis zum Ende ausgeführt
10	J2 wird wieter bis zum Ende ausgeführt. Alle anderen Jobs erden aufgrund ihrer Priorität weiter ausgeführt.

Regeln SPCP:

Annahmen:

Die einmal den Jobs zugeteilte Priorität ist statisch.

Die von den Jobs benötigten Ressourcen sind vor dem Start bekannt.

Die Priority Ceiling $\pi(R)$ jeder Ressource ist die Priorität des höchstprioren Jobs, der diese Ressource benötigt.

Die aktuelle Priority Ceiling Π_s des Systems entspricht der höchsten Priorität der Priority Ceiling $\pi(R)$ der Ressource, die gerade von Jobs benutzt werden.

Wird keine Ressource benutzt so wird $\Pi_s = \Omega$ gesetzt, wobei Ω eine Priorität ist, die niederiger als die realen Prioritäten ist.

1.Berechnung der akuellen Ceiling im System:

Sind alle Ressourcen frei, so gilt $\Pi_s = \Omega$. Π_s wird immer geupdatet, wenn eine Ressource belegt, oder freigegeben wird.

2.Scheduling Regel:

Nachdem der Job released ist, wird er solange blockiert, bis seine zugeortnete Priorität höher ist als die Ceiling des Systems.

Zu jeder Zeit werden alle Jobs, die nicht blockiert sind, entsprechend ihrer zugeordneten Priorität nach einem prioritätenbasierten preemtiven Scheduling ausgeführt.

$\pi_{Job} \leq \hat{\pi}(Ceiling) \rightarrow$ Job wird blockiert

$\pi_{Job} > \hat{\pi}(Ceiling) \rightarrow$ Ready

3.Ressourcen Zuteilung:

Wenn ein Job eine Ressource anfrdert, so wird ihm diese Ressource zugeordnet.

Blockierzeiten bei statischer Priorität:

Job	R1	R2	R3
J1	1	2	0
J2	0	9	3
J3	8	7	0
J4	6	5	4
Π(R)	J1	J1	J2

alle niederprioren Jobs

$\prod(R_k) \geq \prod_i$

Blockierzeiten bei PCP und SPCP:

$b_i = \max_{j,k}\{CS_{j,k} \mid j > i \quad \text{and} \quad \Pi(R_k) \geq \pi_i\}$

J1:

alle signifikanten Stellen $(\Pi(R_k) \geq \pi_i)$ der niederprioren Jobs:

$b_1 = \max(0,9,8,7,6,5) = 9$

J2:

alle Ressourcen zu betracheten, da die Bedingung $\Pi(R_k) \geq \pi_i$ auch für R3 gilt:

$b_2 = \max(8,7,0,6,5,4) = 8$

J3:

wie oben:

$b_3 = \max(6,5,4) = 6$

J4:

$b_4 = 0$

Blockierzeit bei NPCS:

$b_i(rc) = \max_{i+1 \leq k \leq n}(c_k) = \max_{i+1 \leq k \leq n}(\text{Kritischer Bereich})$

J1:

$b_1 = \max(0,9,3,8,7,0,6,5,4) = 9$

J2:

$b_2 = \max(8,7,0,6,5,4) = 8$

J3:

wie oben:

$b_3 = \max(6,5,4) = 6$

J4:

$b_4 = 0$

Blockierzeit bei PIP:

$bt_i = \sum_{j=i+1}^{n} \max_k\{CS_{j,k} \mid \Pi(R_k) \geq \pi_i\} \rightarrow$ waagrecht

$br_i = \sum_{k=1}^{m} \max_{j>i}\{CS_{j,k} \mid \Pi(R_k) \geq \pi_i\} \rightarrow$ senkrecht

$b_i = \min(bt_i, br_i)$

bt_1:

$\max(0,9) + \max(8,7) + \max(6,5) = 23$

br_1:

$\max(0,8,6) + \max(9,7,5) = 17$

bt_2:

$\max(8,7,0) + \max(6,5,4) = 14$

br_2:

$\max(8,6) + \max(7,5) + \max(0,4) = 19$

bt_3:

$\max(6,5,4) = 6$

br_3:

$6 + 5 + 4 = 15$

bt_4:

$bt_4 = br_4 = 0$

Somit ergeben sich die folgenden Blockierzeiten für die Jobs:

$b_1 = \min(23,17) = 17$

$b_2 = \min(14,19) = 14$

$b_3 = \min(6,15) = 6$

$b_4 = 0$

Blockierzeiten bei dynamischer Priorität:

Blockierzeit bei NPCS:

$b_i = \max_{i+1 \leq k \leq n}(CS_k)$

Blockierzeit bei PCP:

J1: $b_1 = \max(0,9,3,8,7,0,6,5,4) = 9$

J2: $b_2 = \max(1,2,0,8,7,0,6,5,4) = 8$

J3: $b_3 = \max(1,2,0,0,9,3,6,5,4) = 9$

J4: $b_4 = \max(1,2,0,0,9,3,8,7,0) = 9$

Echtzeitnachweis bei statischer Priorität:

Jobs	p_i	e_i	d_i	R1	R2	R3
J1	30	3	30	1	2	0
J2	40	12	40	0	9	3
J3	70	15	70	8	7	0
J4	100	15	100	6	5	4

$$\forall T_i / J_i : \frac{b_i}{\min(D_i, p_i)} + \sum_{k=1}^{i} \frac{e_k}{\min(D_k, p_k)} \le i * \left(2^{\frac{1}{i}} - 1\right)$$

$b_1 = 9, \quad b_2 = 8, \quad b_3 = 6, \quad b_4 = 0$

$$J1: \quad \frac{9}{30} + \frac{3}{30} = 0.4 \le 1$$

$$J2: \quad \frac{8}{40} + \left(\frac{3}{30} + \frac{12}{40}\right) = 0.6 \le 0.828$$

$$J3: \quad \frac{6}{40} + \left(\frac{3}{30} + \frac{12}{40} + \frac{15}{70}\right) = 0.7 \le 0.78$$

$$J4: \quad 0 + \left(\frac{3}{30} + \frac{12}{40} + \frac{15}{70} + \frac{15}{100}\right) = 0.7642 \le 0.757$$

$$\square_i \square\square = e_i + b_i + \sum_{k=1}^{i} \frac{\square}{p_k} * e_k \qquad 0 \square\square \le \min(D_i, p_i)$$

$$\square^1 = 15 + 0 + \left(\frac{15}{30} * 3 + \frac{15}{40} * 12 + \frac{15}{70} * 15\right) = 45$$

$$\square^2 = 15 + 0 + \left(\frac{45}{30} * 3 + \frac{45}{40} * 12 + \frac{45}{70} * 15\right) = 60$$

$$\square^3 = 15 + 0 + \left(\frac{60}{30} * 3 + \frac{60}{40} * 12 + \frac{60}{70} * 15\right) = 60$$

D□mi□ e□ □ib□i□□ie m□ im□□e □e□k□n□□ei□□J4 □□: 60 □ De□□□ine □ 10

Echtzeitnachweis bei dynamischher Priorität:

$$\forall T_i / J_i : \frac{b_i}{\min(D_i, p_i)} + \sum_{k=1}^{i} \frac{e_k}{\min(D_k, p_k)} \le 1$$

$b_1 = 9, \quad b_2 = 8, \quad b_3 = 9, \quad b_4 = 9$

$$J1: \frac{9}{30} + \left(\frac{3}{30} + \frac{12}{40} + \frac{15}{70} + \frac{15}{100}\right) = 1.06! \le 1$$

$$J2: \frac{8}{40} + \left(\frac{3}{30} + \frac{12}{40} + \frac{15}{70} + \frac{15}{100}\right) = 0.96 \le 1$$

$$J3: \frac{9}{70} + \left(\frac{3}{30} + \frac{12}{40} + \frac{15}{70} + \frac{15}{100}\right) = 0.89 \le 1$$

$$J4: \frac{9}{100} + \left(\frac{3}{30} + \frac{12}{40} + \frac{15}{70} + \frac{15}{100}\right) = 0.85 \le 1$$

keine □□□ □é□!!!

□e□□□□é□ie □e□□□en□ei□n□□□e□□n□im□e□□ □□ei□min:□□□□

$$\square(\square) = \sum_{i=1}^{n} \frac{\square \quad D_i + p_i \quad \square}{p_i} \square(e_i + b_i)$$