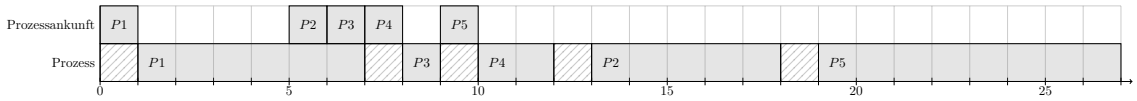
GSS Abgabe 04

Carolin Konietzny, Paul Bienkowski, Julian Tobergte, Oliver Sengpiel

3. Juni 2015

1.1 a) Ausführungsreihenfolge:



Wahl des ersten Prozesses:

Prozesse in Warteschleife	Wartezeit	Bediengüte
P1	0	1

Wahl des zweiten Prozesses:

Prozesse in Warteschleife	Wartezeit	Bediengüte
P2	2	$\frac{7}{5}$
P3	1	2
P4	0	1

Wahl des dritten Prozesses:

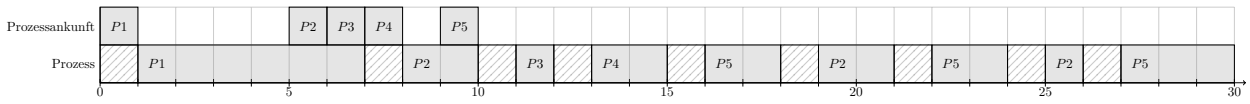
Prozesse in Warteschleife	Wartezeit	Bediengüte
P2	5	$\frac{10}{5}$
P4	4	$\frac{6}{2}$
P5	0	1

Wahl des vierten Prozesses:

Prozesse in Warteschleife	Wartezeit	Bediengüte
P2	7	$\frac{12}{5}$
P5	3	$\frac{11}{8}$

Wahl des fünften Prozesses:
Es bleibt nur noch P5 übrig.

b) Roundrobin:



Hier sind keine Berechnungen nötig, da die Prozesse in der Reihenfolge in die Warteschlange hinzugefügt werden, in der sie ankommen, und sie in dieser Reihenfolge abgearbeitet/getauscht werden.

2 a) Man kann die anteilige Rechenzeit der Prozesse in ihrer jeweiligen Periode ermitteln. Damit ergibt sich, wie lange ein Prozess bei beliebig vielen Perioden den Prozessor belegt. Für unsere Prozesse A1 bis A3 ergibt sich folgendes:

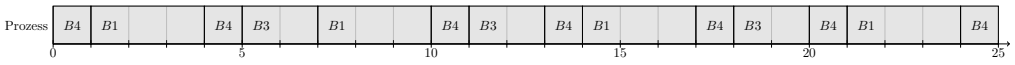
$$P(A_1) = \frac{1}{4} \quad P(A_2) = \frac{3}{7} \quad P(A_3) = \frac{1}{3}$$

Summiert man diese Anteile auf, erhält man einen Wert über 1, das heißt, wir benötigen mehr Rechenzeit, als der eine Prozessor hergibt. Daher ist kein Scheduling möglich:

$$P(A_1) + P(A_2) + P(A_3) \approx 1.0119 > 1$$

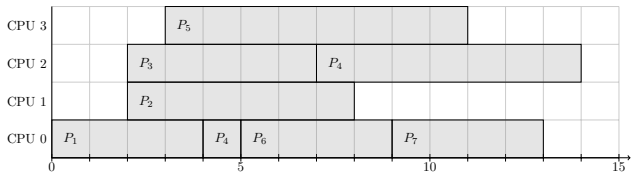
Die Erklärung hierfür ist, dass man zwar mit gutem Scheduling eine gewisse Anzahl an Perioden durchführen kann, dabei aber niemals Rechenzeit „übrig" bleibt, und man sogar irgendwann eine „Rechenzeitschuld" aufgebaut hat, die zu groß ist, als dass man alle Prozesse in ihrer jeweiligen Periode unterbringen könnte.

b) ii) **Rate-montonotic scheduling**

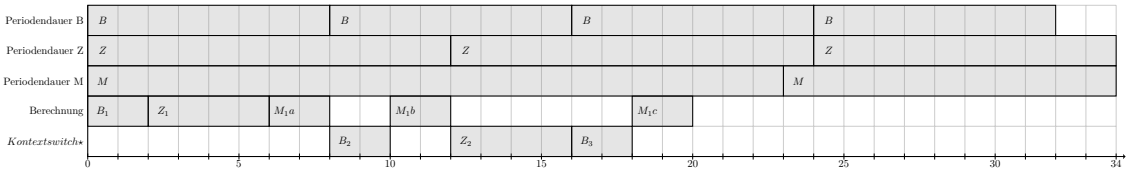


Es lässt sich feststellen, dass $B2$ nicht ein einziges Mal laufen kann. Dies liegt daran, dass es die geringste Priorität (längste Periode) hat, und andere Prozesse gleichzeitig oder kurz vorher wieder bereit werden.

c) **Time-Sharing**



3 a) Zeitskala: $\frac{t}{5}$



⋆ : Während der Bearbeitung von M1 werden Prozessen mit höherer Priorität Rechenzeit zugewiesen, welche „dazwischengeschoben" werden, dh. die Mutexlocks von dem vorherig laufenden Prozess bleiben erhalten, in diesem Fall M1. Dies bedeutet aber immernoch eine sequentielle Ausführung.

Problematisch wird das ganze wenn ein mittelhoch Priorisierter Prozess (Z) Rechenzeit erhält, der dann von einem hochpriorisierten Prozess (B) widerum abgelöst werden soll. Dabei kann nämlich dann der Mutexlock auf einen Wert, auf den M und B zugreifen nicht von M aufgelöst werden, weil M zu diesem Zeitpunkt ja garnicht aktiv ist. Daher würden erst alle anderen Prozess durchlaufen, die höhere Priorität als M haben, bevor B endlich zur Berechnung kommen darf. Dies hat dann bei zeitkritischen Laufplänen die Folge, dass B zu spät rechnen kann.