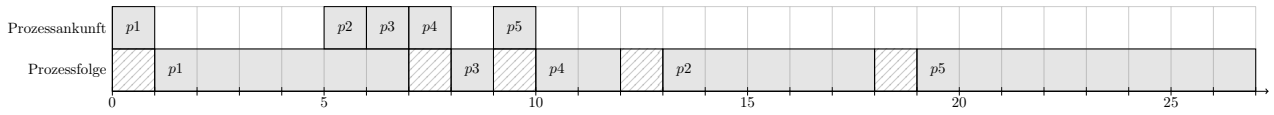
GSS-Übungsblatt 3

Alexander Timmermann, Jannis Krämer

1 Scheduling-Algorithmen

a) Prozessabfolge:



Im ersten Auswahlschritt ist nur $p1$ verfügbar und wird deshalb ausgeführt. Nach dem Abschluss von $p1$ enthält unsere Warteschlange folgende Prozesse mit den entsprechenden Bediengüten:

Prozess	**Wartedauer**	**Bediengüte**
$p2$	2	$\frac{7}{5} = 1{,}4$
$p3$	1	2
$p4$	0	1

Folglich wird $p3$ gewählt, geladen und ausgeführt.

Im nächsten Schritt haben wir folgende Warteschlange:

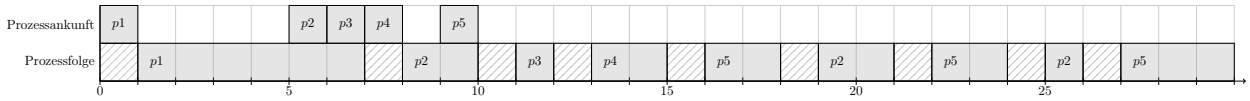
Prozess	**Wartedauer**	**Bediengüte**
$p2$	5	2
$p4$	3	$\frac{5}{2} = 2{,}5$
$p5$	0	1

Es wird $p4$ ausgewählt. Nachdem $p4$ beendet ist haben wir noch folgende Prozesse in der Warteschlange:

Prozess	**Wartedauer**	**Bediengüte**
$p2$	7	$\frac{12}{5} = 2{,}4$
$p5$	3	$\frac{11}{8} = 1{,}375$

Nachdem dann auch $p2$ ausgeführt wurde bleibt nur noch $p5$ in der Queue und wird zuletzt ausgeführt.

b) Prozessabfolge:



Bei Round Robin wird jeder Prozess in der Reihenfolge in die Warteschlange eingefügt in der er bereitsteht. In der gleichen Reihenfolge wird auch abgearbeitet bzw. getauscht.

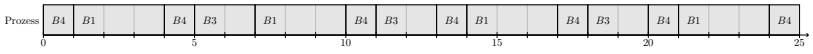
2 Echtzeit- und Multiprozessor-Scheduling

a) Addiert man die Anteile der Periode der Prozesse, der tatsächlich Rechenzeit ist, so kann man ausrechnen wie lange alle Prozesse zusammen bei beliebig vielen Perioden den Prozessor belasten. Ist dieser Wert größer als 1 so ist der Prozessor über 100% der Zeit belegt, was bedeuten würde, dass Scheduling unmöglich wäre. Für die gegebenen 3 Prozesse ergibt sich:

$$P(A_1) + P(A_2) + P(A_3) = \frac{1}{4} + \frac{3}{7} + \frac{1}{3} = \frac{85}{84} > 1$$

Ein Scheduling dieser 3 Prozesse auf nur einem Prozessor ist somit nicht möglich.

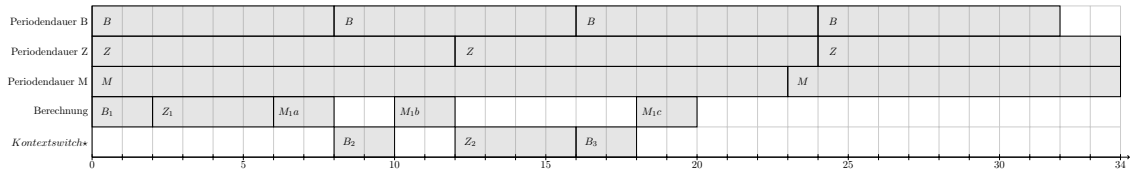
b) ii) Mit Rate Monotone Scheduling ergibt sich folgende Grafik:



Wegen der geringen Priorität von B2, die sich aus der langen Periodendauer und der gleichzeitig geringen Bedienzeitforderung ergibt wird B2 nie ausgeführt. Durch die teils deutlich niedrigeren Periodendauern und den daraus resultierenden höheren Prioritäten steht immer ein Auftrag mit höherer Priorität bereit der statt b2 ausgeführt wird.

3 Prioritätsinversion

a) Es ergibt sich folgende Abbildung:



Während der Bearbeitung von M1 wird B aufgrund seiner höheren Priorität eingeschoben. M1 gibt dabei seine Mutexlocks weiter, obwohl M1 noch nicht alle Daten schreiben konnte. Im Zuge des Bus Management, das B ausführt, benötigt B nämlich auch Zugriff auf auf die Daten von M1. Nachdem B durchgelaufen ist läuft M1 somit weiter. Bevor M1 seinen Task jedoch beenden kann wird er abermals unterbrochen, diesmal jedoch von Z. Z läuft mit mittlerer Priorität, löst M1 somit ab. Z benötigt jedoch keinen Zugriff auf von M1 geschriebene Daten, erhält also auch nicht die Mutexlocks von M1. Nachdem Z nun fertig ist wird er allerdings von B abgelöst, nicht von M, da B die höchste Priorität besitzt. B1 hat nun keinen Zugriff auf Ms Daten, da M nicht aktiv ist und somit B auch keine Mutexlocks übertragen kann. B kann deshalb nicht zuende rechnen und muss warten bis M seine Daten fertig geschrieben hat. Da jedoch zuerst alle Prozesse mit höherer Priorität als M ausgeführt werden, kann die Ausführung B unter Umständen sehr lange verhindert sein und ein zeitkritisches System somit zum Absturz bringen. Der Computer der Pathfinder-Mission beispielweise führte automatisch einen Neustart, welcher mit Datenverlust verbunden war, aus wenn der Bus Management Task (B) zu lange nicht ausgeführt wurde.