

# Tutorübung Grundlagen: Betriebssysteme und Systemsoftware

Moritz Beckel

München, 18. November 2022

Freitag 10:15-12:00 Uhr Raum ([00.11.038](#))

Zulip-Stream <https://zulip.in.tum.de/#narrow/stream/1295-GBS-Fr-1000-A>

Unterrichtsmaterialien findet ihr hier:

<https://home.in.tum.de/~beckel/gbs>

Lösungen wurden von mir selbst erstellt. Es besteht keine Garantie auf Korrektheit.

Betriebsart	Schedulingstrategien	Optimierungsziele
Batch-Systeme	<ul style="list-style-type: none"> <li>• First-Come-First-Served non-preemptive</li> <li>• Shortest Job First (SJF) non-preemptive</li> <li>• Shortest Remaining Time Next (SRTN) preemptive</li> </ul>	<ul style="list-style-type: none"> <li>• Durchsatz (Anzahl Aufträge max.)</li> <li>• CPU-Belegung (max.)</li> <li>• Ausführungszeit (Wartezeit + Rechenzeit min.)</li> </ul>
Interaktive Systeme	<ul style="list-style-type: none"> <li>• Round Robin (RR) preemptive</li> <li>• Priority Scheduling preemptive</li> </ul>	<ul style="list-style-type: none"> <li>• Antwortzeit (min.)</li> <li>• Proportionalität (Schneller Mausklick vs. langsamer Download)</li> </ul>
Echtzeit Systeme	<ul style="list-style-type: none"> <li>• Earliest Deadline First (EDF) (non)-preemptive</li> <li>• Rate-Monotonic Scheduling (RMS) preemptive</li> </ul>	<ul style="list-style-type: none"> <li>• Deadlines einhalten</li> <li>• Vorhersagbarkeit der Rechendauer</li> <li>• Kein Verhungern</li> </ul>

# Aufgabe 1

Prozess	Thread	Startzeit	Rechenzeit
$P_1$	$K_1$	0	13
$P_1$	$K_2$	3	3
$P_1$	$K_3$	20	2
$P_2$	$U_1$	2	8
$P_2$	$U_2$	9	5
$P_2$	$U_3$	12	4

## Kernel-Scheduler:

- Round-Robin
- 5 Einheiten Zeitquantum
- Prozesse in der Reihenfolge ihrer Startzeit abgearbeitet
- Kernel-Scheduler benötigt eine Zeiteinheit
- Kontextwechsel kostet eine Zeiteinheit
- initiale Aktivität nicht berücksichtigt

## User-Level-Scheduler:

- Round-Robin
- Thread nach zwei Zeiteinheiten oder beim Terminieren die Kontrolle an den User-Level-Scheduler abgibt
- Threads in der Reihenfolge ihrer Ankunft bearbeitet
- Sobald ein neuer Thread hinzu kommt, gibt der laufende Thread die CPU ab und der neue Thread bekommt sie zugewiesen
- User-Level-Scheduling kostet keine Zeiteinheiten
- unabhängig vom Kernel-Scheduler

# Aufgabe 1

[illegible]

# Aufgabe 1

Prozess	Thread	Startzeit	Rechenzeit
$P_1$	$K_1$	0	13
$P_1$	$K_2$	3	3
$P_1$	$K_3$	20	2
$P_2$	$U_1$	2	8
$P_2$	$U_2$	9	5
$P_2$	$U_3$	12	4

	Scheduling																																																	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
$P_1, K_1$	×	×	×	×	×	—	—	—	—	—	—	—	—	—	—	—	—	—	×	×	×	×	×	—	—	—	—	—	—	—	—	—	—	×	×	×														
$P_1, K_2$				—	—	—	—	—	—	—	—	—	—	×	×	×																																		
$P_1, K_3$																					—	—	—	—	—	—	—	—	—	—	—	×	×																	
$P_2, U_1$			—	—	—	—	—	×	×	—	—	×	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	×	×	—	—	—	—	—	—	—	—	—	—	—	—	—	×	×	—	—	×			
$P_2, U_2$										×	×	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	×	—	—	—	—	—	—	—	—	—	—	×	—	—	—	—	—	×			
$P_2, U_3$													—	—	—	—	—	—	—	—	—	—	—	—	—	×	×	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	×	×						
ed/Disp						×	×						×	×				×							×	×						×	×														×			

## Aufgabe 2

Seit der Linux Kernel Version 2.6.23 wird der sogenannte **Completely-Fair-Scheduler (CFS)** verwendet, um zu entscheiden, welcher Prozess als nächstes an die CPU gebunden werden soll. Im Folgenden werden wir uns mit diesem etwas vertrauter machen.

## Aufgabe 2

Der CFS unterstützt **40 statische Prioritätsstufen**, mit denen festgelegt werden kann, welchen **Anteil jeder Prozess an der vergeben CPU-Zeit** bekommen soll.

Die Prioritäten des CFS werden indirekt durch den **niceness Wert eines Prozesses** bestimmt. Dieser Wert kann unter Linux (z.B. durch die Programme top oder htop) dynamisch an Prozesse vergeben werden und variiert **zwischen -20 und +19**, wobei **kleinere Werte eine höhere Priorität**, also einen geringeren Grad an „Nettigkeit“ bedeuten.

## Aufgabe 2

Jedem **nice**ness Wert ist ein **Gewicht w** zugeordnet, das im Linux Kernel (kernel/sched/core.c) im folgenden Array vorgegeben wird:

```
const int sched_prio_to_weight[40] = {  
    /* -20 */      88761,      71755,      56483,      46273,      36291,  
    /* -15 */      29154,      23254,      18705,      14949,      11916,  
    /* -10 */       9548,       7620,       6100,       4904,       3906,  
    /*  -5 */       3121,       2501,       1991,       1586,       1277,  
    /*   0 */       1024,        820,        655,        526,        423,  
    /*   5 */        335,        272,        215,        172,        137,  
    /*  10 */        110,         87,         70,         56,         45,  
    /*  15 */         36,         29,         23,         18,         15,  
};
```



## Aufgabe 2

Es wird eine **time slice TS** berechnet, die sowohl die Anzahl als auch die Prioritäten aller aktuell aktiver Prozesse auf dem System berücksichtigt. Bei der Berechnung der time slice TS wird der Wert der **targeted latency TL** mit in Betracht genommen. Dieser Wert stellt ein **Intervall** dar, in dem **alle Prozesse mindestens einmal die CPU bekommen** sollen. Der TL Wert ist **fest vorgegeben**. Berechnung der time slice TL:

$$TS_i = TL * \frac{w_i}{\sum_{j=1}^n w_j}$$

## Aufgabe 2

Zusätzlich zum vergebenen Zeitquantum muss der Scheduler entscheiden, **welcher Prozess als nächstes an die CPU** gelassen werden soll.

**Normalerweise** würde man hierfür vermutlich den Prozess auswählen, der bis dato **am wenigsten Rechenzeit  $rt$**  (real runtime) bekommen hat. Da wir hier allerdings berücksichtigen müssen, dass **Prozesse mit höherer Priorität grundsätzlich länger an die CPU** gelassen werden, verwenden wir stattdessen eine **virtual runtime  $vt$** . Diese **verrechnet die real runtime** zusätzlich **mit der Priorität** des entsprechenden Prozesses.

$$vt_{i_{new}} = vt_{i_{old}} + \frac{w_0}{w_i} (rt_{i_{new}} - rt_{i_{old}}) \text{ mit } w_0 = 1024$$

$$TS_i = TL * \frac{w_i}{\sum_{j=1}^n w_j}$$

$$vt_{i_{new}} = vt_{i_{old}} + \frac{1024}{w_i} (rt_{i_{new}} - rt_{i_{old}})$$

Prozess	Rechenzeit	Niceness	Weight
1	20	0	1024
2	25	-5	3121
3	5	1	820
4	3	18	18
5	33	-10	9548

t	rt <sub>1</sub>	TS <sub>1</sub>	vt <sub>1</sub>	rt <sub>2</sub>	TS <sub>2</sub>	vt <sub>2</sub>	rt <sub>3</sub>	TS <sub>3</sub>	vt <sub>3</sub>	rt <sub>4</sub>	TS <sub>4</sub>	vt <sub>4</sub>	rt <sub>5</sub>	TS <sub>5</sub>	vt <sub>5</sub>
0															
4															
15															

## CFS

- targeted latency **TL 50 ms**
- TS und vt auf ganze Zahlen aufgerundet
- Ankunftszeiten  **$\vec{a} = (0, 0, 0, 0, 0)$**
- Anfangswerte für rt und vt haben Wert 0

$$TS_i = TL * \frac{w_i}{\sum_{j=1}^n w_j}$$

$$vt_{i_{new}} = vt_{i_{old}} + \frac{1024}{w_i} (rt_{i_{new}} - rt_{i_{old}})$$

Prozess	Rechenzeit	Niceness	Weight
1	20	0	1024
2	25	-5	3121
3	5	1	820
4	3	18	18
5	33	-10	9548

t	rt <sub>1</sub>	TS <sub>1</sub>	vt <sub>1</sub>	rt <sub>2</sub>	TS <sub>2</sub>	vt <sub>2</sub>	rt <sub>3</sub>	TS <sub>3</sub>	vt <sub>3</sub>	rt <sub>4</sub>	TS <sub>4</sub>	vt <sub>4</sub>	rt <sub>5</sub>	TS <sub>5</sub>	vt <sub>5</sub>
0	0	4* <sup>1</sup>	0	0	11	0	0	3	0	0	1	0	0	33	0
4	4	4	4	0	11	0	0	3	0	0	1	0	0	33	0
15	4	4	4	11	11	4* <sup>2</sup>	0	3	0	0	1	0	0	33	0

## CFS

- targeted latency TL 50 ms
- TS und vt auf ganze Zahlen aufgerundet
- Ankunftszeiten  $\vec{a} = (0, 0, 0, 0, 0)$
- Anfangswerte für rt und vt haben Wert 0

$$TS_i = TL * \frac{w_i}{\sum_{j=1}^n w_j}$$

$$vt_{i_{new}} = vt_{i_{old}} + \frac{1024}{w_i} (rt_{i_{new}} - rt_{i_{old}})$$

Prozess	Rechenzeit	Niceness	Weight
1	20	0	1024
2	25	-5	3121
3	5	1	820
4	3	18	18
5	33	-10	9548

## CFS

- targeted latency **TL 50 ms**
- TS und vt auf ganze Zahlen aufgerundet
- Ankunftszeiten  $\vec{a} = (0, 0, 0, 0, 0)$
- Anfangswerte für rt und vt haben Wert 0

t	rt <sub>1</sub>	TS <sub>1</sub>	vt <sub>1</sub>	rt <sub>2</sub>	TS <sub>2</sub>	vt <sub>2</sub>	rt <sub>3</sub>	TS <sub>3</sub>	vt <sub>3</sub>	rt <sub>4</sub>	TS <sub>4</sub>	vt <sub>4</sub>	rt <sub>5</sub>	TS <sub>5</sub>	vt <sub>5</sub>
0	0	4* <sub>1</sub>	0	0	11	0	0	3	0	0	1	0	0	33	0
4	4	4	4	0	11	0	0	3	0	0	1	0	0	33	0
15	4	4	4	11	11	4* <sub>2</sub>	0	3	0	0	1	0	0	33	0
18							3	3	4						
19										1	1	57			
52													33	33	4
63	15	11	15												
77				25	32	9									
79							5	23	7						
84	20	50	20												
86										3	50	171			

# Aufgabe 5 Midterm 2021

- a) Die Prozesse **P1, P2, P3 und P4** wurden nacheinander auf einem System mit **einem Rechenkern** anhand eines **unbekannten Scheduling Verfahrens** ausgeführt. Das folgende Gantt Diagramm beschreibt deren Ausführungsreihenfolge. Für **welches Optimierungsziel** wurde die Scheduling Reihenfolge vermutlich optimiert? Nennen Sie das Optimierungsziel und beschreiben Sie es kurz.

Betriebsart	Schedulingstrategien	Optimierungsziele
Batch-Systeme	<ul style="list-style-type: none"> <li>• First-Come-First-Served non-preemptive</li> <li>• Shortest Job First (SJF) non-preemptive</li> <li>• Shortest Remaining Time Next (SRTN) preemptive</li> </ul>	<ul style="list-style-type: none"> <li>• Durchsatz (Anzahl Aufträge max.)</li> <li>• CPU-Belegung (max.)</li> <li>• Ausführungszeit (Wartezeit + Rechenzeit min.)</li> </ul>
Interaktive Systeme	<ul style="list-style-type: none"> <li>• Round Robin (RR) preemptive</li> <li>• Priority Scheduling preemptive</li> </ul>	<ul style="list-style-type: none"> <li>• Antwortzeit (min.)</li> <li>• Proportionalität (Schneller Mausklick vs. langsamer Download)</li> </ul>
Echtzeit Systeme	<ul style="list-style-type: none"> <li>• Earliest Deadline First (EDF) (non)-preemptive</li> <li>• Rate-Monotonic Scheduling (RMS) preemptive</li> </ul>	<ul style="list-style-type: none"> <li>• Deadlines einhalten</li> <li>• Vorhersagbarkeit der Rechendauer</li> <li>• Kein Verhungern</li> </ul>

# Aufgabe 5 Midterm 2021

	Rechenzeit	Ankunftszeit	IO Interrupt vorhanden	Länge IO Interrupt	IO nach
$P_1$	3	1	Ja	4	2
$P_2$	2	5	Nein	-	-
$P_3$	4	1	Nein	-	-
$P_4$	9	1	Nein	-	-

	0	5	10	15	20	25	30
$P_1$	X	X					
$P_2$							
$P_3$	-	-	-	X	X	X	X
$P_4$	-	-	-	-	-	-	-
Scheduler		X		X		X	



# Aufgabe 5 Midterm 2021

	Rechenzeit	Ankunftszeit	IO Interrupt vorhanden	Länge IO Interrupt	IO nach
$P_1$	3	1	Ja	4	2
$P_2$	2	5	Nein	-	-
$P_3$	4	1	Nein	-	-
$P_4$	9	1	Nein	-	-

	0		5			10			15			20			25			30		
$P_1$		X	X				-	-	X											
$P_2$						-	-	-	-	-	X	X								
$P_3$		-	-	-	X	X	X	X												
$P_4$		-	-	-	-	-	-	-	-	-	-	-	X	X	X	X	X	X	X	
Scheduler				X				X		X		X								

- Scheduling optimiert auf Durchsatz, es wird versucht möglichst viele Aufträge in möglichst geringer Zeit zu erledigen

# Aufgabe 5 Midterm 2021

- c) Im Folgenden werden alle Prozesse anhand des **preäemptiven Scheduling** Verfahrens **Shortest Remaining Time Next** geschedult. Ergänzen Sie alle fehlenden Angaben.
- **Scheduling und Dispatching kostet jeweils eine Zeiteinheit.**
  - Jedes Mal wenn ein **Prozess rechenbereit** wird, wird der **Scheduler** für eine Zeiteinheit **aktiv**.
  - **Scheduler und Dispatcher** können **auch zu anderen Zeiten aktiv** werden.
  - In jedem Prozess kann **maximal eine IO-Unterbrechung** auftreten.
  - Vernachlässigen Sie den initialen Scheduler/Dispatcher Aufruf.

# Aufgabe 5 Midterm 2021

	Rechenzeit	Ankunftszeit	IO Interrupt vorhanden	Länge IO Interrupt	IO nach
$P_1$			Ja		
$P_2$					
$P_3$		4			

	0	5	10	15	20	25	30
$P_1$							
$P_2$							
$P_3$							
Scheduler		x	x x	x	x	x	
Dispatcher			x	x	x		

# Aufgabe 5 Midterm 2021

	Rechenzeit	Ankunftszeit	IO Interrupt vorhanden	Länge IO Interrupt	IO nach
$P_1$	4	3	Ja	1	3
$P_2$	2	10	Ja	2	1
$P_3$	8	4	Nein		

