Lehrstuhl für Connected Mobility
School of Computation, Information and Technology
Technische Universität München

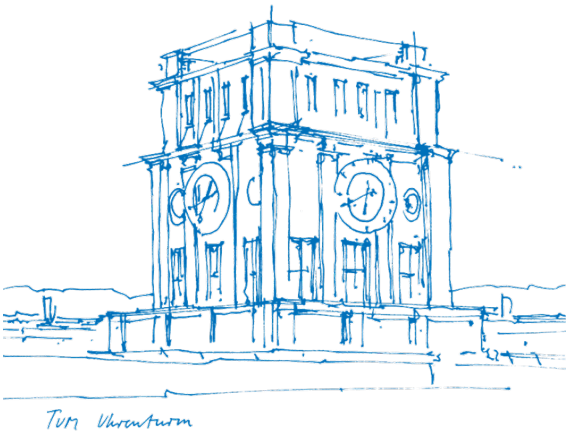
Grundlagen: Betriebssysteme und Systemsoftware

Tutorübung

Mario Delic

Lehrstuhl für Connected Mobility
School of Computation, Information and Technology
Technische Universität München

Übungswoche 4



Scheduling
Ziele

- **Generell**: $\rightarrow$ Fairness, Balance
- **Batch-Systeme**: $\rightarrow$ Durchsatz, Ausführungszeit, CPU Belegung
- **Interaktive Systeme**: $\rightarrow$ Antwortzeit, Proportionalität
- **Echtzeit**: $\rightarrow$ Deadlines, Vorhersagbarkeit

Scheduling
Scheduling-Ebenen

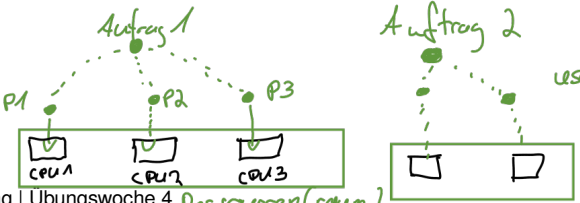
Aufrufhäufigkeit

- **short-term scheduler (CPU scheduler**) → Teilt Prozessen CPU zu
- medium-term scheduler → Teil vom swapping-Mechanismus
- long-term scheduler (Job Scheduler) → Scheduled die ready-Queue

Scheduling
Varianten

A1 & 2 teilen sich die Zeit auf der Ressource CPU1

- **Time sharing** (auch: time slicing) → Mehrere Aufträge teilen sich durch timeslices eine Ressource gleichzeitig
- **Space sharing** (auch: space slicing) → Zusammenhängende Prozesse werden als ein Auftrag zusammengefasst und bekommen eine Ressource bis zu Terminierung
- **Gang Scheduling** → Kombination: Zusammenhängede Prozesse werden als ein Auftrag zusammengefasst (Gang), und teilen sich über timeslices mit anderen Gangs eine Ressource → ≈ „Gangs timeshoren den Ressourcenraum"



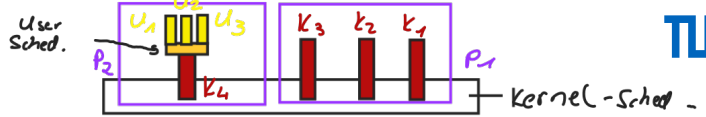
user... A1 & 2 bekommen einen Ressourcenraum, den sich ihre Prozesse bis Terminierung teilen

Scheduling
Real World Constraints

- **NUMA Systeme** → Systeme, auf denen Speicherzugriffszeiten für Prozesse unterschiedlich sind: Sinnvolle Platzierung der Prozesse auf jeweils geeigneten Prozessoren 'nahe' am relevanten Speicher (Verteilungsproblem)
- **Energie** → In Systemen mit begrenzter Energie muss sparsam und effizient gescheduled werden z.B. Handys
- **Hitze** → Prozesse, die viel Wärme generieren sollten nicht auf Prozessoren gescheduled werden, die gerade überhitzen
- **Bottlenecks** → Umgang mit Prozesses die viel und/oder andere Prozesse blockieren?
- **Cache Lokalität** → Sinnvolle Nutzung des Cache spart aufwendige Speicherzugriffsoperationen und kann z.B. den Durchsatz erhöhen. Vorhersage/Approximierung der Cache-Nutzung eines Programms?

Aufgabe 1

User+Kernel Scheduling



Modellierung von Round Robin Scheduling für Kernel- als auch User-Level-Scheduler.

Kernel-Level Scheduler:

- Zeitquantum = **5 Zeiteinheiten**
- Zyklische Abarbeitung nach Ankunftszeit
- Aktivierung des Kernel-Schedulers = **1 Zeiteinheit**
- Kontextwechsel (Aktivierung des Dispatchers) = **1 Zeiteinheit**

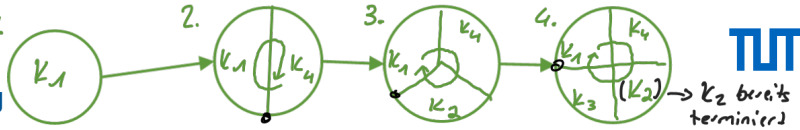
User-Level Scheduler:

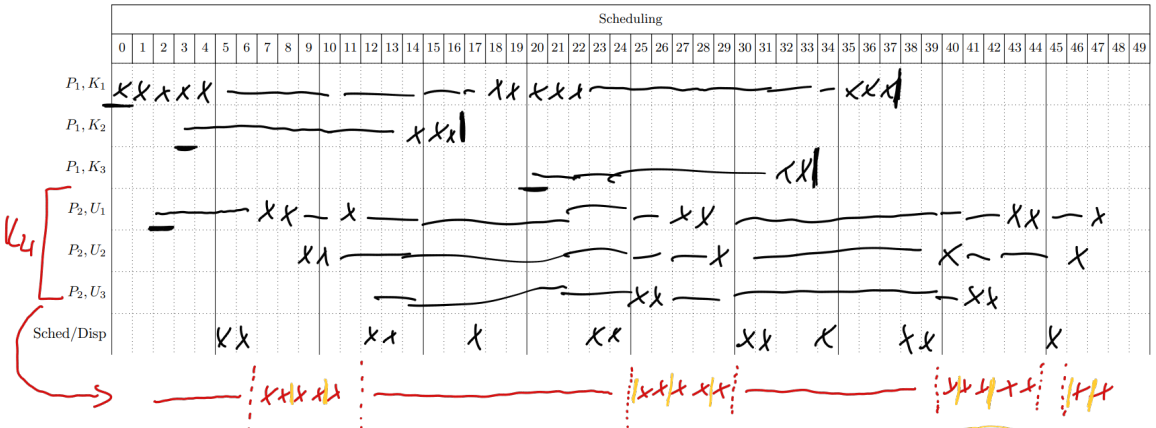
- Zeitquantum = **2 Zeiteinheiten**
- Zyklische Abarbeitung nach Ankunftszeit
- Aktivierung des User-Schedulers = vernachlässigbar
- Preemptive bei Ankunft neuer Threads

Kernel und User-Level-Scheduler schedulen ihre eigenen Threads **unabhängig voneinander**!

Aufgabe 1

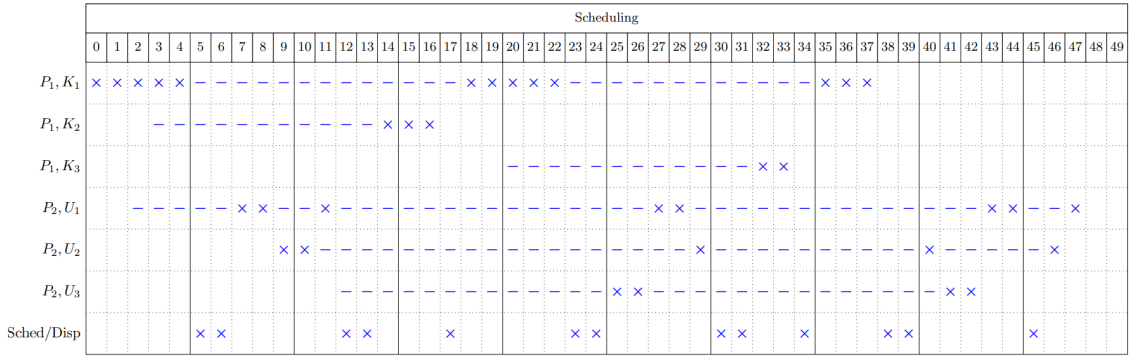
User+Kernel Scheduling





Aufgabe 1

User+Kernel Scheduling



Aufgabe 2
Linux CFS

- Completely Fair Scheduler seit Version 2.6.23 (Jahr 2007) im Einsatz.
- **Niceness**: Wert im Intervall [-20;19]. Sagt aus, wie sehr ein Prozess seine CPU-Zeit an andere Prozesse 'verschenkt'.
  $\hookrightarrow$ 0: neutral, 19: very nice, -20: very mean.
- Geringere niceness bedeutet höhere Priorität - und umgekehrt.
- Niceness < 0 setzen erfordert root-Rechte!

- Jedem niceness Wert ist ein Gewicht zugeordnet (siehe Array $sched\_prio$)
- Dynamische Zeitscheiben/timeslices statt statische!
- **rt** (real runtime) = Rechenzeit des Prozesses.
- **vt** (virtual runtime) = mit Priorität verrechnete rt.

Aufgabe 2

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

Formel für TimeSlice von i (zur Vereinfachung mit TL = 50):

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

Formel für virtual runtime von i:

$$vt_{i\_new} = vt_{i\_old} + \frac{w_0}{w_i}(rt_{i\_new} - rt_{i\_old}), \text{ mit } w_0 = 1024 \tag{2}$$

$vt_3 \rightarrow 0 + \frac{1024}{820}(11 - 0) \approx \frac{1}{3} \cdot 11 \approx 4$

Aufgabe 2

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

Formel für TimeSlice von i (zur Vereinfachung mit TL = 50):

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

→ Prozesse die Terminiert sind scheiden aus dieser Summe aus!

Formel für virtual runtime von i:

$$vt_{i\_new} = vt_{i\_old} + \frac{w_0}{w_i}(rt_{i\_new} - rt_{i\_old}), \text{ mit } w_0 = 1024 \quad (2)$$

	P_1			P_2			P_3			P_4			P_5		
t	rt_1	TS_1	vt_1	rt_2	TS_2	vt_2	rt_3	TS_3	vt_3	rt_4	TS_4	vt_4	rt_5	TS_5	vt_5
0	0	4*1	0	0		0	0		0	0		0	0		0
4	4	4	4	0	11	0	0	2.	0	0		0	0		0
15	4	4	4	11	11	4*2	0	3	0	0		0	0		0
18	4.						3	3. 3	4 5	0	1	0	0		0
19											...			...	

1. Wähle den Prozess der als nächstes rechnen soll (hier: geringste vt_i)
2. Berechne TS_i mittels Formel: $TS_3 = 50 \cdot \frac{820}{1024 + 3121 + 820 + 18 + 3548} \approx 3$
3. Addiere TS_i auf $rt_i \rightarrow rt_{i\_new} = rt_{i\_old} + TS_i$

wichtig: nicht die maximale Rechenzeit des Prozesses überschreiten

	P_1			P_2			P_3 1.			P_4			P_5		
t	rt_1	TS_1	vt_1	rt_2	TS_2	vt_2	rt_3	TS_3	vt_3	rt_4	TS_4	vt_4	rt_5	TS_5	vt_5
0	0	4*1	0	0		0	0		0	0		0	0		0
4	4	4	4	0	11	0	0	2.	0	0		0	0		0
15	4	4	4	11	11	4*2	0	3	0	0		0	0		0
18	4.						3	3. 3	4 5.	0	1 …	0	0	…	0
19															

Erläuterungen zum Vorgehen anhand Prozess/Schritt 3:

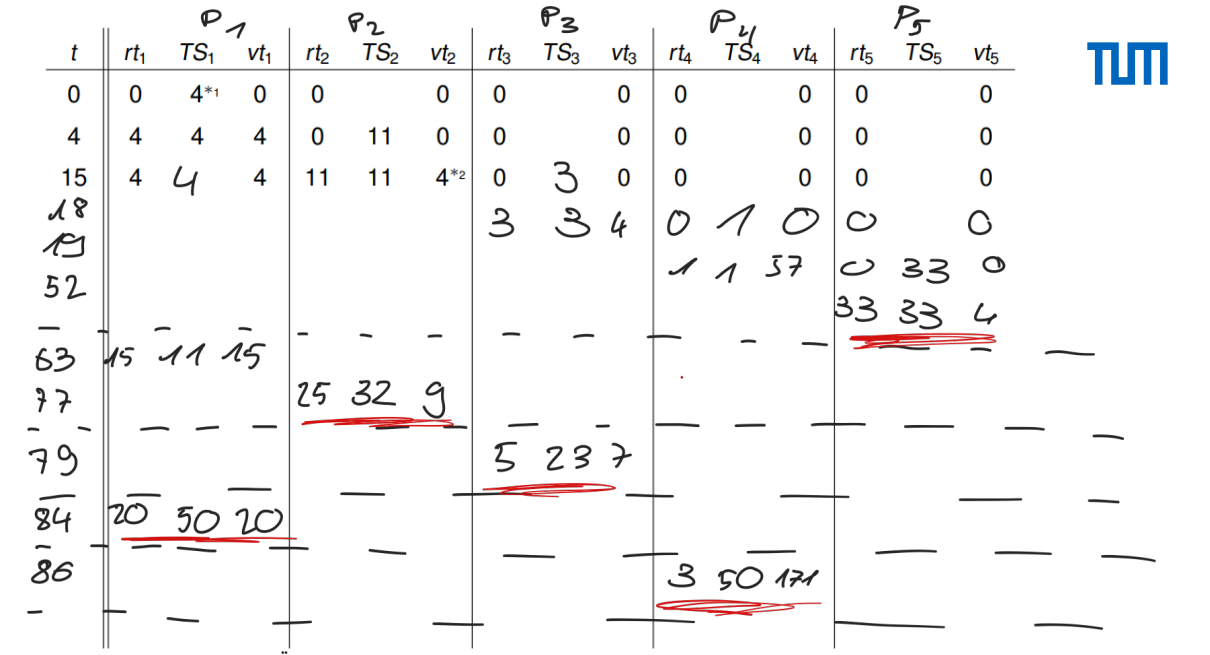
4. t um die tatsächliche Rechenzeit des Prozesses in diesem Schritt Updaten

$\rightarrow t_{new} = t_{old} + (rt_{new} - rt_{old})$

5. vt_i mit Formel berechnen: $vt_{3\_new} = vt_{3\_old} + \frac{1024}{820} \cdot (3-0) = 0 + \frac{1024}{820} \cdot 3 \approx 4$

→ Fertig!

1. Nächsten Prozess mit geringster vt_i wählen …


P1
P2
P3
P4
P5
t
rt1 TS1 vt1
rt2 TS2 vt2
rt3 TS3 vt3
rt4 TS4 vt4
rt5 TS5 vt5
0
4
15
18
19
52
63
77
79
84
86
4*1
4*2
15 11 15
25 32 9
5 23 7
20 50 20
3 50 171