

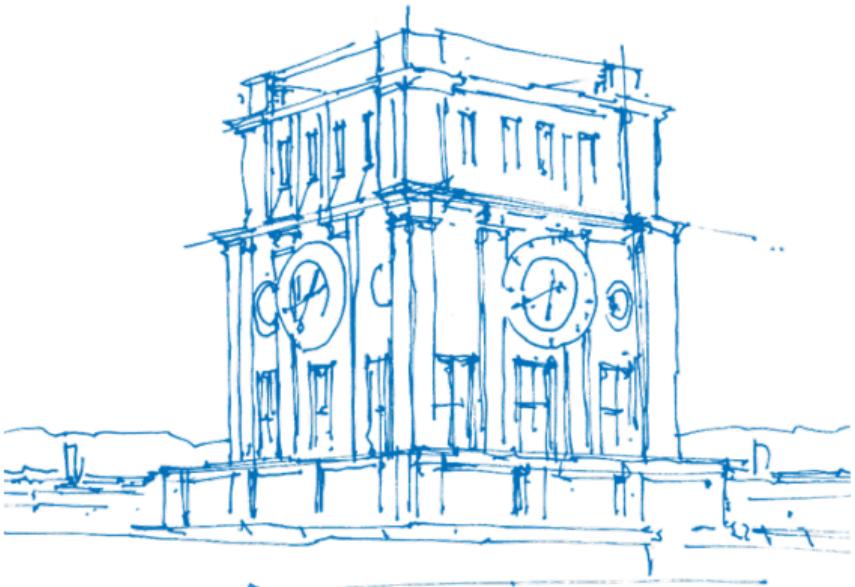
Grundlagen: Betriebssysteme und Systemsoftware

Tutorübung

Mario Delic

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

Übungswoche 4



TUM Uhrenturm

Scheduling

Ziele

- **Generell:** → Fairness, Balance
- **Batch-Systeme:** → Durchsatz, Ausführungszeit, CPU Belegung
- **Interaktive Systeme:** → Antwortzeit, Proportionalität
- **Echtzeit:** → 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

- **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: Zusammenhängende Prozesse werden als ein Auftrag zusammengefasst (Gang), und teilen sich über timeslices mit anderen Gangs eine Ressource

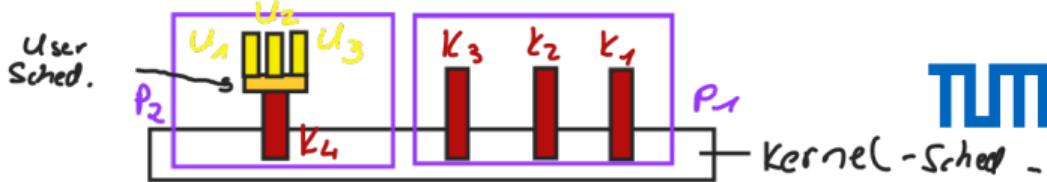
Scheduling

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 geschedulet werden, die gerade überhitzen
- **Bottlenecks** → Umgang mit Prozessen 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-Schedulers.

Kernel-Level Scheduler:

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

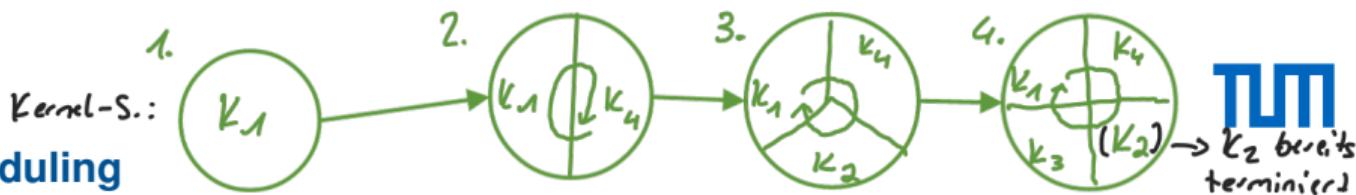
User-Level Scheduler:

- Zeitscheibe = **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'.
→ 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
- 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):

$$TS_i = TL * \frac{w_i}{\sum_{j=1}^n w_j} \xrightarrow{\text{Prozesse die Terminiert sind scheiden aus dieser Summe aus!}} \quad (1)$$

Formel für virtual runtime von i:

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

t	P_1		P_2		P_3		P_4		P_5						
	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	0	0	0	0
4	4	4	4	0	11	0	0	2.	0	0	0	0	0	0	0
15	4	4	4	11	11	4*2	0	3	0	0	0	0	0	0	0
18	4.						3	3.	3	1	0	0	0	0	0
19							3	3.	4						

Erläuterungen zum Vorgehen anhand Prozess/Schritt 3:

- Wähle den Prozess der als nächstes Rechnen soll
(hier: geringste vt_i)
- Berechne TS_i mittels Formel: $TS_3 = 50 \cdot \frac{820}{1024 + 3121 + 820 + 18 + 3548} \approx 3$

- Addiere TS_i auf $rt_i \rightarrow rt_{i_new} = rt_{i_old} + TS_i$
wichtig: nicht die maximale Rechenzeit des Prozesses überschreiten

t	P_1			P_2			P_3			P_4			P_5		
	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	0	0	0	0
4	4	4	4	0	11	0	0	2.	0	0	0	0	0	0	0
15	4	4	4	11	11	4*2	0	3	0	0	0	0	0	0	0
18	4.						3	3.	3	0	1	0	0	0	0
19							3	3.	3	4

Erläuterungen zum Vorgehen anhand Prozess/Schritt 3:

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

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

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

\rightarrow Fertig!

1. Nächsten Prozess mit geringster vt_i wählen ...

t	P_1			P_2			P_3			P_4			P_5		
	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	0	0	0	0
4	4	4	4	0	11	0	0	0	0	0	0	0	0	0	0
15	4	4	4	11	11	4*2	0	3	0	0	0	0	0	0	0
18							3	3	4	0	1	0	0	0	0
19										1	1	57	0	33	0
52													33	33	4
63	15	11	15	-	-	-	-	-	-	-	-	-	-	-	-
77				25	32	9				.					
79	-	-	-				-	5	23	7					
84	20	50	20	-			-								
86	-	-	-	-			-	3	50	171					
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Ansonsten...

- Anmeldung für die Midterm seit 14.11 offen!