

Betriebssysteme 1

Foliensatz D, Kap. 3 (Scheduling)

Prof. Dr. Hans-Georg Eßer

Sommersemester 2022
v3.0 – 06.05.2021

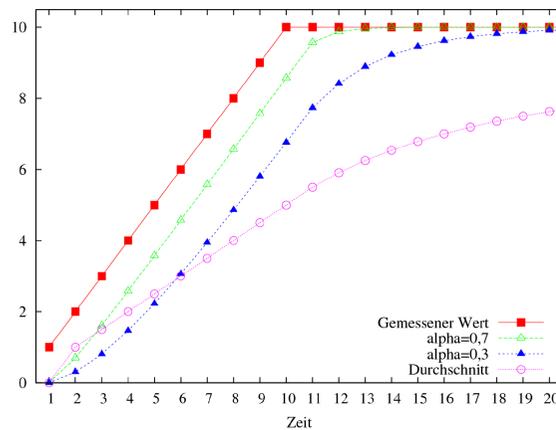
- Vorsicht (auch für Prüfung): SJF und SRT werden oft verwechselt.
- „SJF < SRT“
 - (SJF ist i.d.R. schlechter als SRT, weil es keine Unterbrechungen kennt)
 - dieser Vergleich gilt auch lexikalisch (Eselsbrücke)

Einfache Verfahren

- FCFS
- SJF und SRT → Burst-Dauer-Prognose?
 - Mittelwert
 - Exponentieller Durchschnitt

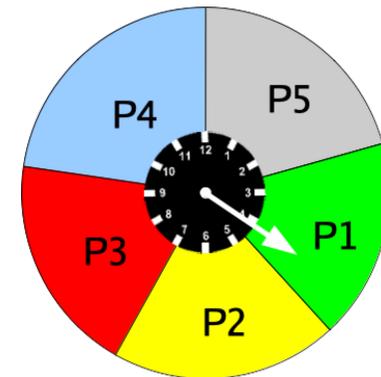
$$S_{n+1} = \alpha T_n + (1 - \alpha) S_n$$

α : Gewicht zwischen 0 und 1



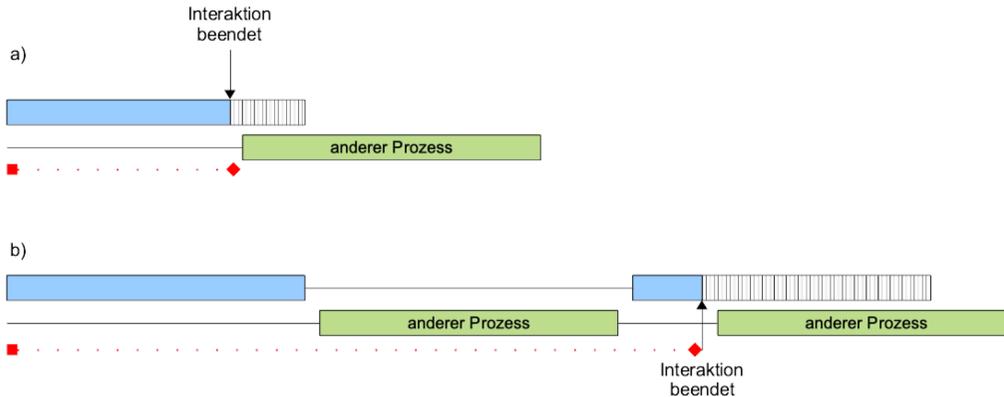
Round Robin

- Wie FCFS – aber mit Unterbrechungen
- Alle bereiten Prozesse in einer Warteschlange (quantum, time slice) zuordnen
- Jedem Thread eine Zeitscheibe (quantum, time slice) zuordnen
- Ist Prozess bei Ablauf der Zeit-scheibe noch aktiv, dann:
 - Prozess verdrängen (→ Zustand „bereit“)
 - Prozess ans Ende der Warteschlange hängen
- Wie groß soll das Quantum sein?



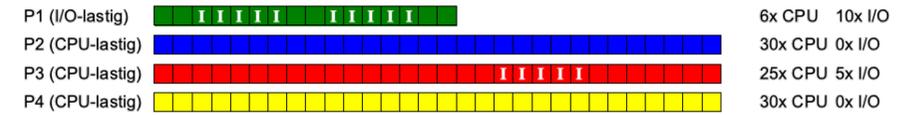
Round Robin: Wahl des Quantums

- Oft: Quantum q etwas größer als typische Zeit, die das Bearbeiten einer Interaktion benötigt

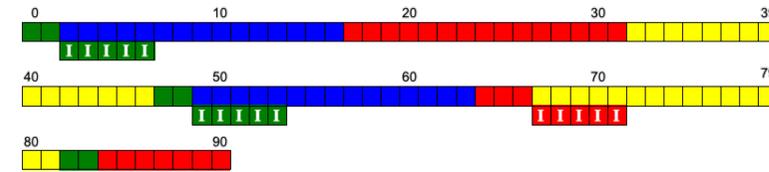


Round Robin: I/O- vs. CPU-lastig

Ideler Verlauf (wenn jeder Prozess exklusiv läuft)



Ausführreihenfolge mit Round Robin, Zeitquantum 15:

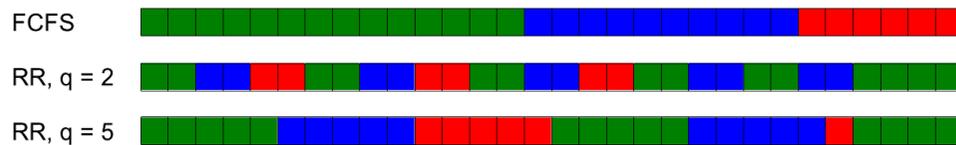


Prozess	CPU-Zeit	I/O-Zeit	Summe	Laufzeit	Wartezeit *)
P1	6	10	16	84	68
P2	30	0	30	64	34
P3	25	5	30	91	61
P4	30	0	30	82	52

*) im Zustand bereit, nicht blockiert!

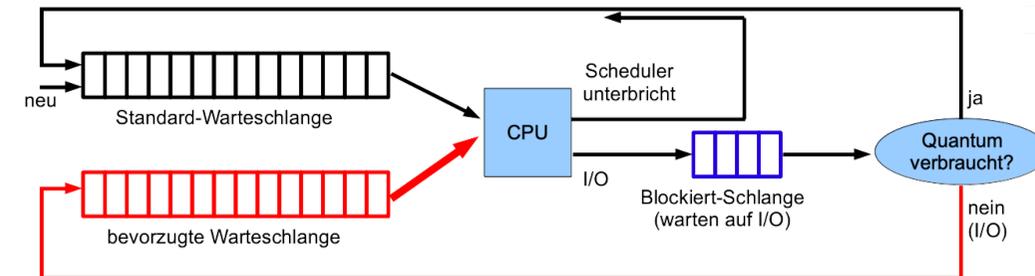
Round Robin: Beispiel

- Szenario: Drei Prozesse
 - FCFS (einfache Warteschlange, keine Unterbrechung)
 - Round Robin mit Quantum 2
 - Round Robin mit Quantum 5

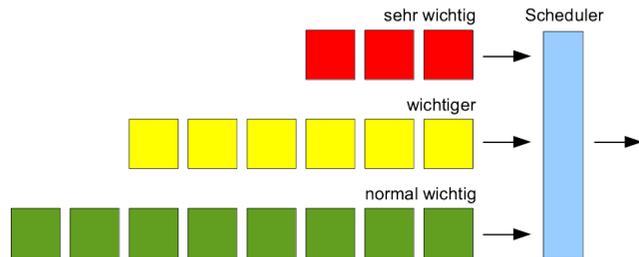


Virtual Round Robin

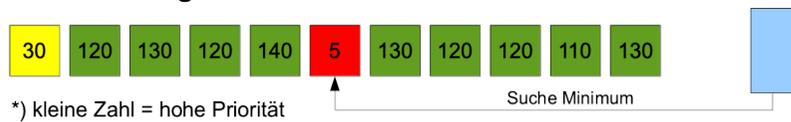
- **Beobachtung:** RR unfair zu I/O-lastigen Prozessen
 - CPU-lastige nutzen ganzes Quantum,
 - I/O-lastige nur einen Bruchteil
- **Lösungsvorschlag: VRR**



- Mehrere Warteschlangen (Prioritätsklassen)



- Warteschlange mit individuellen Prioritätswerten



Fragen: Skript, S. 86 / 87

- 13. Scheduling-„Rechenaufgabe“

Zeit	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9 (10+)	0 1 2 3 4 5 6 7 (20+)
FCFS	P P P P P P P P P P	Q Q	
SJF			
SRT			
RR(6)			
RR(12)			

- 14. Lotterie-Scheduler

```
int random1 (int range) {
    static int last = 0; // wird beim ersten (!) Aufruf auf 0 initialisiert.
    last++;
    if (last > range)
        last = 1;
    return last;
}

int random2 (int range) {
    return 1; // Ich ziehe immer die Eins.
}
```

- **Prioritätsinversion:**

- Prozess hoher Priorität ist blockiert (benötigt ein Betriebsmittel)
- Prozess niedriger Priorität besitzt dieses Betriebsmittel, wird aber vom Scheduler nicht aufgerufen (weil es wichtigere Prozesse gibt)
- Beide Prozesse kommen nie dran, weil immer Prozesse mittlerer Priorität laufen
- Ausweg: Aging

- **Aging:**

- Priorität eines Prozesses, der bereit ist und auf die CPU wartet, wird regelmäßig erhöht
- Priorität des aktiven Prozesses und aller nicht-bereiten (blockierten) Prozesse bleibt gleich
- Ergebnis: Lange wartender Prozess erreicht irgendwann ausreichend hohe Priorität, um aktiv zu werden