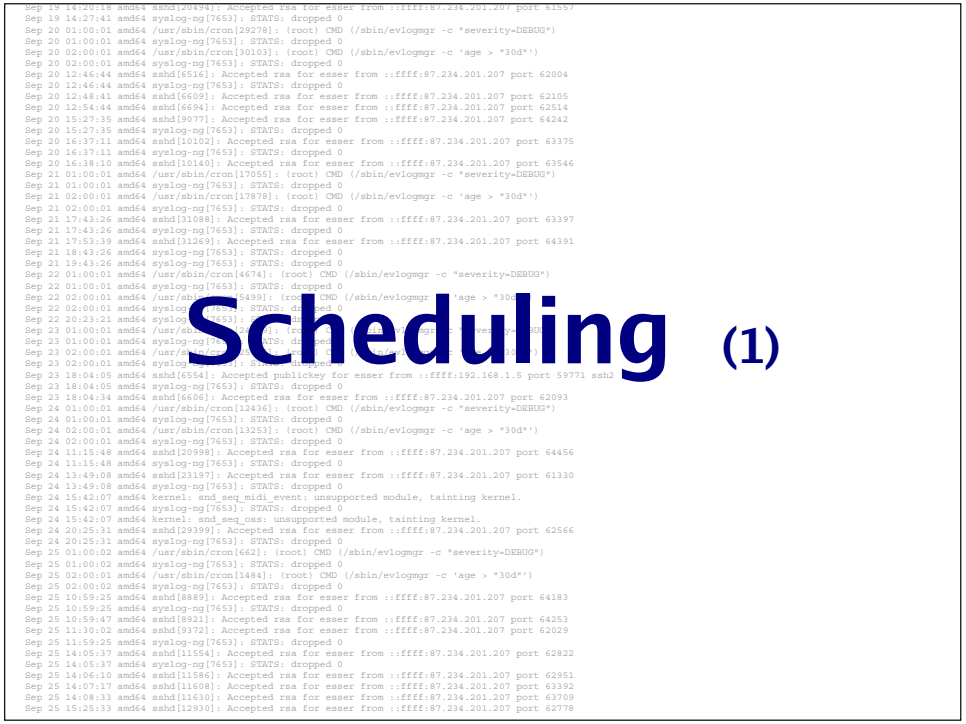


Scheduling (1)

Einführung

Gliederung

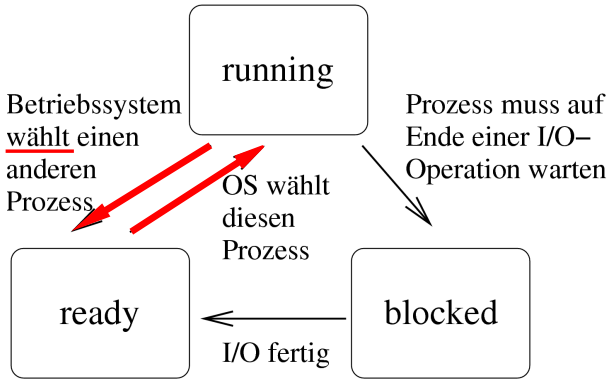
- Was ist Scheduling? Motivation
- Kooperatives / präemptives Scheduling
- CPU- und I/O-lastige Prozesse
- Ziele des Scheduling (abhängig vom BS-Typ)
- Standard-Scheduling-Verfahren
- Praxis: Linux-Scheduler

Scheduling – worum geht es?

- Multitasking: Mehrere Prozesse konkurrieren um Betriebsmittel
- Betriebssystem verwaltet die Betriebsmittel
- Rechenzeit auf dem Prozessor
- Scheduler entscheidet:
  Welchen Prozess wann ausführen?
- Ausführreihenfolge entscheidend für Gesamt-Performance des Betriebssystems

Scheduling: Prozess auswählen

Zustandsübergänge



Wann wird Scheduler aktiv?

- Neuer Prozess entsteht (fork)
- Aktiver Prozess blockiert wegen I/O-Zugriff
- Blockierter Prozess wird bereit
- Aktiver Prozess endet (exit)
- Prozess rechnet schon zu lange
- Interrupt tritt auf

Unterbrechendes Scheduling

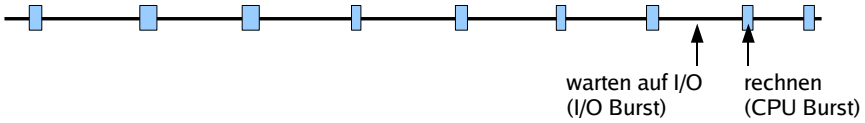
Prozess-Unterbrechung möglich?

- **Kooperatives Scheduling:**
  - Prozess rechnet solange, wie er will; bis zum nächsten I/O-Aufruf oder bis `exit()`
  - Scheduler wird nur bei Prozess-Blockieren oder freiwilliger CPU-Aufgabe aktiv
- **Präemptives (unterbrechendes) Scheduling:**
  - Timer aktiviert regelmäßig Scheduler, der neu entscheiden kann, „wo es weiter geht“

Prozesse: I/O- oder CPU-lastig

I/O-lastig:

- Prozess hat zwischen I/O-Phasen nur kurze Berechnungsphasen (CPU)



CPU-lastig:

- Prozess hat zwischen I/O-Phasen lange Berechnungsphasen

Häufige Prozesswechsel?

Faktoren

- **Zeit für Kontext-Switch:** Scheduler benötigt Zeit, um Prozesszustand zu sichern
  ➞ verlorene Rechenzeit
- **Wartezeit der Prozesse:** Häufigere Wechsel erzeugen stärkeren Eindruck von Gleichzeitigkeit

Ziele des Scheduling (1)

Aus Anwendersicht

- **[A1] Ausführdauer:** Wie lange läuft der Prozess insgesamt?
- **[A2] Reaktionszeit:** Wie schnell reagiert der Prozess auf Benutzerinteraktion?
- **[A3] Deadlines** einhalten
- **[A4] Vorhersehbarkeit:** Gleichartige Prozesse sollten sich auch gleichartig verhalten, was obige Punkte angeht
- **[A5] Proportionalität:** „Einfaches“ geht schnell

Ziele des Scheduling (2)

Aus Systemsicht

- **[S1] Durchsatz:** Anzahl der Prozesse, die pro Zeit fertig werden
- **[S2] Prozessorauslastung:** Zeit (in %), die der Prozessor aktiv war
- **[S3] Fairness:** Prozesse gleich behandeln, keiner darf „verhungern“
- **[S4] Prioritäten** beachten
- **[S5] Ressourcen** gleichmäßig einsetzen

[A1] Ausführdauer

Wie viel Zeit vergeht vom Programmstart bis zu seinem Ende?

- n Prozesse p_1 bis p_n starten zum Zeitpunkt t_0 und sind zu den Zeitpunkten t_1 bis t_n fertig
- Durchschnittliche Ausführdauer: $1/n \cdot \sum_i (t_i - t_0)$
- Abhängig von konkreten Prozessen; Berechnung nur für Vergleich verschiedener Scheduling-Verfahren sinnvoll

[A2] Reaktionszeit

Wie schnell reagiert das System auf Benutzereingaben?

- Benutzer drückt Taste, klickt mit Maus etc. und wartet auf eine Reaktion
- Wie lang ist die Zeit zwischen Auslösen des Interrupts und Aktivierung des Prozesses, der die Eingabe auswertet?
- Toleranz bei langen Wartezeiten gering; schon 2-4 Sekunden kritisch, darüber inakzeptabel

[A3] Deadlines

Hält das System Deadlines ein?

- Realtime-Systeme: besondere Ansprüche
- Prozesse müssen in vorgegebener Zeit ihre Aufgaben erledigen, also ausreichend und rechtzeitig Rechenzeit erhalten
- Wie oft werden Deadlines nicht eingehalten?
- Optimiere (prozentualen) Anteil der eingehaltenen Deadlines

[A4] Vorhersehbarkeit

Ähnliches Verhalten ähnlicher Prozesse?

- Intuitiv: Gleichartige Prozesse sollten sich auch gleichartig verhalten, d. h.
  - Ausführdauer und Reaktionszeit immer ähnlich
  - Unabhängig vom sonstigen Zustand des Systems
- Schwierig, wenn das System beliebig viele Prozesse zulässt ➞ Beschränkungen?

[A5] Proportionalität

Vorgänge, die „einfach“ sind, werden schnell erledigt

- Es geht um das (evtl. falsche) Bild, das Anwender sich von technischen Abläufen machen
- Benutzer akzeptiert Wartezeit eher, wenn er den zugrunde liegenden Vorgang als komplex einschätzt

[S1] Durchsatz

Terminierende Prozesse

- Anzahl der Prozesse, die pro Zeiteinheit (z. B. pro Stunde) fertig werden, sollte hoch sein
- Misst, wie viel Arbeit erledigt wird
- Abhängig von konkreten Prozessen; Berechnung nur für Vergleich verschiedener Scheduling-Verfahren sinnvoll

[S2] Prozessorauslastung

CPU immer gut beschäftigt halten

- Anteil der Taktzyklen, in denen die CPU nicht „idle“ war
- Interessanter Faktor, wenn Rechenzeit sehr teuer ist (kommerzielles Rechenzentrum)

[S3] Fairness

Alle Prozesse haben gleiche Chancen

- Jeder Prozess sollte mal drankommen (kein „Verhungern“, engl. *process starvation*)
- Keine großen Abweichungen bei den Wartezeiten und Ausführdauern
- Falls Prozess-Prioritäten:
  ➡ „manche sind gleicher“

[S4] Prioritäten

Verschieden wichtige Prozesse auch verschieden behandeln

- Prioritätsklassen: Prozesse mit hoher Priorität bevorzugt behandeln
- Dabei verhindern, dass nur noch Prozesse mit hoher Priorität laufen (und alles andere steht)

[S5] Ressourcen-Balance

„BS verwaltet die Betriebsmittel...“

- Grundidee des BS: alle Ressourcen gleichmäßig verteilen und gut auslasten
- CPU-Scheduler hat auch Einfluss auf (un)gleichmäßige Auslastung der I/O-Geräte
- Prozesse bevorzugen, die wenig ausgelastete Ressourcen nutzen wollen

Anforderungen an das Betriebssystem (1)

Drei Kategorien

- Stapelverarbeitung
- Interaktives System
- Echtzeitsystem

Immer wichtig:

- S3 Fairness
- S4 Prioritäteneinsatz
- S5 Ressourcen-Balance

Anforderungen an das Betriebssystem (2)

Stapelverarbeitung

- S3 Fairness
- S4 Prioritäteneinsatz
- S5 Ressourcen-Balance
- S1 Durchsatz
- A1 Ausführdauer
- S2 Prozessor-Auslastung

Anforderungen an das Betriebssystem (3)

Interaktives System

- S3 Fairness
- S4 Prioritäteneinsatz
- S5 Ressourcen-Balance
- A2 Reaktionszeit
- A5 Proportionalität

Anforderungen an das Betriebssystem (4)

Echtzeitsystem

- S3 Fairness
- S4 Prioritäteneinsatz
- S5 Ressourcen-Balance
- A3 Deadlines
- A4 Vorhersehbarkeit

Scheduler für Stapelverarbeitung (Batch-Systeme)

Stapelverarbeitung (Batch)

Eigenschaften der Stapelverarbeitung

- Nicht interaktives System (keine normalen Benutzerprozesse)
- Jobs werden über Job-Verwaltung abgesetzt; System informiert über Fertigstellung
- Typische Aufgaben: Lange Berechnungen, Kompiliervorgänge

Stapelverarbeitung (Batch)

Historisch:

- Batch-Betrieb mit Lochkarten
- Programm-Code und Daten auf Lochkarten
- Keine I/O (keine Geräte außer Kartenleser, Drucker)
- Kartenstapel (engl.: batch) legt Reihenfolge fest
  - Programm-Code von Karten lesen
  - Daten von Karten lesen
  - Berechnung durchführen
  - Ergebnis auf Karten stanzen oder drucken
  - Nächster Job

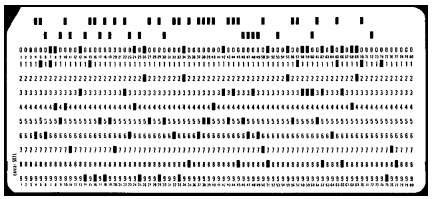


Bild: http://www.fao.org/docrep/X5738E/x5738e0h.htm

Stapelverarbeitung (Batch)

Moderne Batch-Systeme

- Normale Rechner (mit Platten, Netzwerk etc.)
- Kein interaktiver Betrieb (kein Login etc.)
- Job-Management-Tool nimmt Jobs an
- Long term scheduler entscheidet, wann ein Job gestartet wird – evtl. basierend auf Informationen über Ressourcenverbrauch und erwartete Laufzeit des Programms

Stapelverarbeitung (Batch)

Scheduling-Verfahren für Batch-Betrieb

- First Come, First Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time Next
- Priority Scheduling

First Come, First Served (FCFS)

Einfache Warteschlange

- Neue Prozesse reihen sich in Warteschlange ein
- Scheduler wählt jeweils nächsten Prozess in der Warteschlange
- Prozess arbeitet, bis er fertig ist (kooperatives Scheduling)

FCFS-Beispiel

Drei Prozesse mit Rechendauern

T1: 15 Takte
T2: 4 Takte
T3: 3 Takte

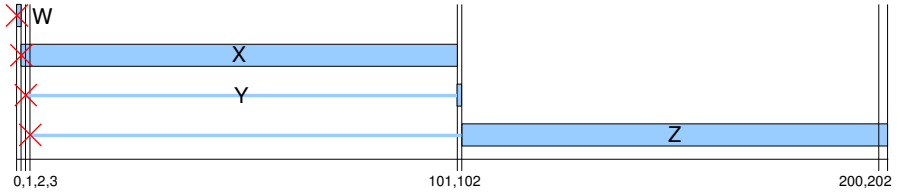
Durchschnittliche Ausführdauer:

a) (15+19+22) / 3 = 18,67
b) (3+ 7+22) / 3 = 10,67
c) (3+18+22) / 3 = 14,33

FCFS: Gut für lange Prozesse

- FCFS bevorzugt lang laufende Prozesse
- Beispiel: 4 Prozesse W, X, Y, Z

Prozess	Ankunftzeit	Service Time T_s (Rechenzeit)	Startzeit	Endzeit	Turnaround T_r (Endzeit-Ankunftzeit)	T_r/T_s
W	0	1	0	1	1	1,00
X	1	100	1	101	100	1,00
Y	2	1	101	102	100	**100,00**
Z	3	100	102	202	199	1,99



FCFS: CPU- vs. I/O-lastige Prozesse

FCFS bevorzugt CPU-lastige Prozesse

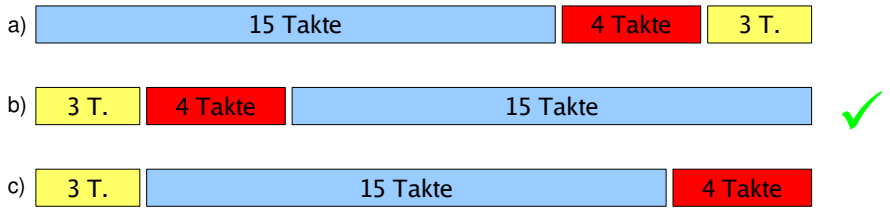
- Während CPU-lastiger Prozess läuft, müssen alle anderen Prozesse warten
- I/O-lastiger Prozess kommt irgendwann dran, läuft nur sehr kurz und muss sich dann wieder hinten anstellen
- Ineffiziente Nutzung sowohl der CPU als auch der I/O-Geräte

Shortest Job First (SJF)

- Keine Unterbrechungen (wie FCFS)
- Nächste Rechendauer (Burst) aller Prozesse bekannt oder wird geschätzt
- Strategie: Führe zunächst den Prozess aus, der am kürzesten laufen wird
- Minimiert die durchschnittliche Laufzeit aller Prozesse
- Prinzip war schon in FCFS-Beispiel erkennbar

SJF-Beispiel

Im Beispiel von der FCFS-Folie:
Ausführreihenfolge b) entspricht SJF



SJF-Eigenschaften

+ [A2] Reaktionszeit: insgesamt deutlich besser, aber Varianz größer (stärkere Ausrutscher), deswegen:
– [A4] Vorhersehbarkeit schlecht

SJF-Eigenschaften

Generelles Problem:
Woher wissen, wie lange die Prozesse laufen?

- Batch-System; Programmierer muss Laufzeit schätzen -> Bei grober Fehleinschätzung: Job abbrechen
- System, auf dem immer die gleichen / ähnliche Jobs laufen -> Statistiken führen
- Interaktive Prozesse: Durchschnitt der bisherigen Burst-Längen berechnen

Ohne diese Information ist dieses Scheduling-Verfahren nur ein theoretisches

Burst-Dauer-Prognose (1)

Einfachste Variante: Mittelwert

$$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i = \frac{1}{n} T_n + \frac{n-1}{n} S_n$$

mit:
T_i: Dauer des i-ten CPU-Burst des Prozess
S_i: Vorausgesagte Dauer des i-ten CPU-Burst
S_1: Vorausgesagte Dauer des 1. CPU-Burst (nicht berechnet)

Burst-Dauer-Prognose (2)

Exponentieller Durchschnitt

$S_{n+1} = \alpha T_n + (1-\alpha) S_n$ α: Gewicht zwischen 0 und 1

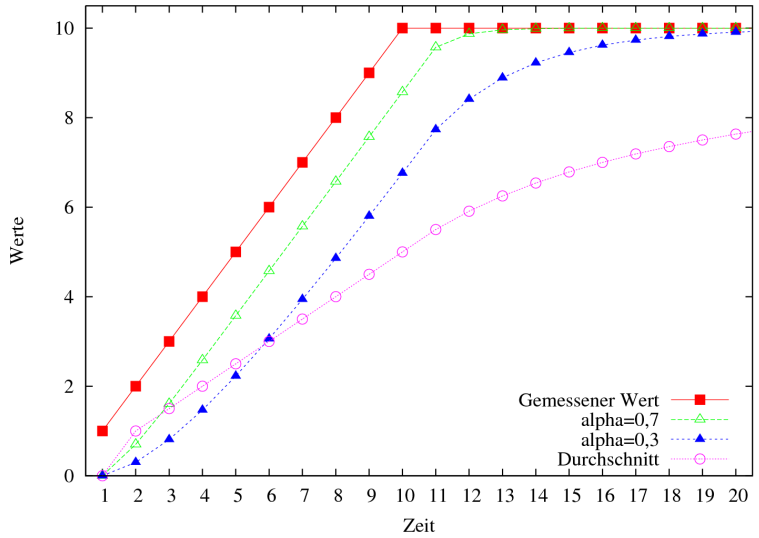
Beispiel: $\alpha = 0{,}8$:

$$S_2 = 0{,}8\, T_1 + 0{,}2\, S_1$$

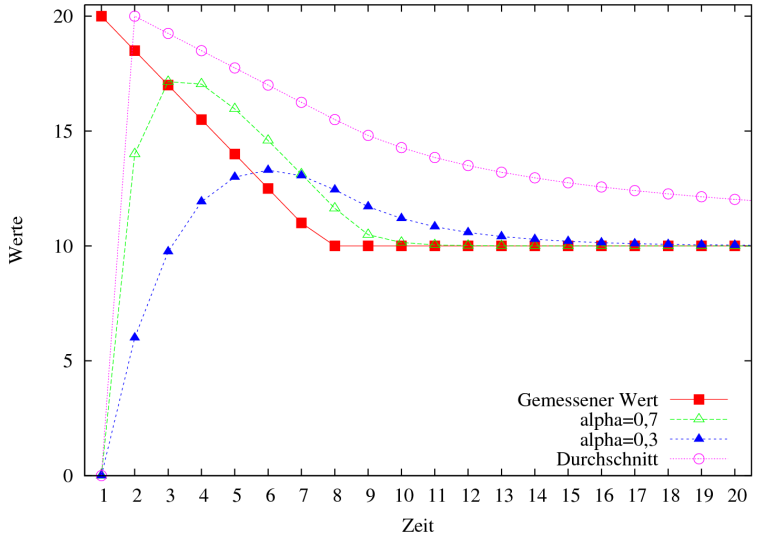
$$S_3 = 0{,}8\, T_2 + 0{,}2\, S_2 = 0{,}8\, T_2 + 0{,}2\,(0{,}8\, T_1 + 0{,}2\, S_1)$$
$$\ldots = 0{,}8\, T_2 + 0{,}16\, T_1 + 0{,}04\, S_1$$

$$S_{n+1} = \sum_{i=0}^{n} (1-\alpha)^{n-i} \alpha\, T_i \quad \text{mit} \quad T_0 := S_1$$

Burst-Dauer-Prognose (3)



Burst-Dauer-Prognose (4)



Shortest Remaining Time (SRT)

- Ähnelt SJF, aber:
- präemptiv (mit Unterbrechungen)
- Regelmäßig Neuberechnung, wie viel Restzeit die Prozesse noch benötigen werden
- Scheduler prüft Reihenfolge immer, wenn ein neuer Job erzeugt wird
- Für kürzeren (auch neuen) Job wird der aktive unterbrochen
- Wie bei SJF gute Laufzeitprognose nötig

SRT-Beispiel

Altes FCFS-Beispiel: SRT unterbricht jetzt X:
Denn Y kommt zwar später, ist aber kürzer

Prozess	Ankunftzeit	Service Time T_s (Rechenzeit)	Startzeit	Endzeit	Turnaround T_r (Endzeit-Ankunftzeit)	T_r/T_s
W	0	1	0	1	1	1,00
X (1)	1	100	1	2 (*)		
Y	2	1	2	3	1	1,00
X (2)			3	102	102-1=101	1,01
Z	3	100	102	202	199	1,99