Lernzettel

Betriebssystemgrundlagen: Prozesse, Threads, Scheduling-Algorithmen, Kontextwechsel

Universität:	Technische Universität Berlin
Kurs/Modul:	Technische Grundlagen der Informatik
Erstellungsdatum:	September 19, 2025

Lernzettel: Betriebssystemgrundlagen: Prozesse, Threads, Scheduling-Algorithmen, Kontextwechsel

(1) Prozesse.
Ein Prozess ist ein laufendes Programm mit eigenem Adressraum, eigener Ressourcenzuordnung (Speicher, Offenbarungen, Dateien) und eigener Ausführungskontext. Er durchläuft typischerweise folgende Zustände: *Neu (New)*, *Bereit (Ready)*, *Ausführend (Running)*, *Wartend/Blockiert (Waiting/Blocked)* und *Beendet (Terminated)*.

Eigenschaften von Prozessen:

- Trennung von Adressräumen (Schutz), wodurch Fehler isoliert bleiben.
- Kontextwechsel: Wechsel des Ausführungskontexts zwischen Prozessen.
- Scheduler als Teil des Betriebssystems koordiniert die Zuteilung von CPU-Zeit.

(2) Threads.
Threads sind die kleinsten Ausführungseinheiten. Mehrere Threads können denselben Adressraum eines Prozesses teilen und so Ressourcen effizient nutzen. Vorteile von Threads: geringerer Overhead beim Kontextwechsel, bessere Ausnutzung von Mehrkernprozessoren. Unterschiede:

- *Userlevel-Threads* oder *Kernel-Threads* (oder Hybridmodelle).
- Threads besitzen eigenen Stack, aber denselben Adressraum wie andere Threads des gleichen Prozesses.

(3) Scheduling-Algorithmen.
Der Scheduler entscheidet, welcher Prozess bzw. welcher Thread die CPU erhält. Unterscheidung in preemptiv vs. nicht-preemptiv, sowie Performance-Kriterien wie Durchsatz, Reaktionszeit, Fairness und Vermeidung von Starvation.

Typen und typische Strategien:

- *FCFS* (First-Come, First-Served) – nicht-preemptiv: einfache Implementierung, aber schlechte Reaktionszeit.
- *SJF* (Shortest Job First) – nicht-preemptiv: geringe mittlere Wartezeit, aber Multiplikator-Skalierbarkeit und Gefahr der Hungern.
- *SRTF* (Shortest Remaining Time First) – preemptiv: reagiert gut, kann Hungern verursachen.
- *Round Robin* (RR) – preemptiv: feste Zeitquantum q; gute Reaktionszeit, Konflikte bei zu kleinem oder zu großem q.
- *Priority Scheduling* – vorrangig, ggf. mit Preemption; kann Hungern verursachen, daher oft Aging (Nachlauf) einsetzen.
- *Multilevel Queue / Multilevel Feedback Queue* – mehrere Ebenen mit unterschiedlichen Regeln; adaptiv.

(4) Kontextwechsel.
Beim Kontextwechsel wird der laufende Kontext des aktuellen Threads/Prozesses gespeichert und der Kontext des nächsten ausgewählt. Wichtige Bestandteile des Kontexts:

- CPU-Register, Programmzähler (PC), Stackpointer (SP), Statusregister.
- ggf. Übersetzungs- bzw. Seitentabellen-Zustände (TLB/MMU) sowie Interrupt-Status.

Der Ablauf eines Kontextwechsels umfasst das Speichern des aktuellen Kontexts, das Laden des nächsten Kontexts sowie das Aktualisieren von Scheduler-Quellen. Diese Operation verursacht Overhead, der umso kleiner sein sollte, je schneller der Kontextwechsel umgesetzt wird.

(5) Beispiel: Vergleich FCFS vs Round Robin (q = 2) für 3 Prozesse.
Angenommene Burst-Zeiten (CPU-Bedarf) in diskreten Zeitscheiben:

$$P1 : 6, \quad P2 \qquad\qquad : 2, \quad P3 : 8$$

FCFS (Reihenfolge P1, P2, P3):

- Ablauf: P1 0-6, P2 6-8, P3 8-16
- Warteschlangen-Wartezeiten W: W(P1)=0, W(P2)=6, W(P3)=8
- Durchschnittliche W-zeit (0+6+8)/3 = 4,67
- Turnaround-Zeiten T: T(P1)=6, T(P2)=8, T(P3)=16; Durchschnitt 10

Round Robin (q = 2):

- Ablauf: P1 0-2, P2 2-4, P3 4-6, P1 6-8, P3 8-10, P1 12-?,

(Kurzfassung der Sequenzen: nach 0-2 P1 bleibt 4 übrig, nach 2-4 P2 beendet, 4-6 P3 mit 6 übrig, 6-8 P1 mit 2 übrig, 8-10 P3 mit 4 übrig, 10-12 P1 beendet, 12-14 P3 mit 2 übrig, 14-16 P3 beendet.)

- Endzeiten: P2 beendet bei 4, P1 beendet bei 12, P3 beendet bei 16
- Turnarounds T: T(P1)=12, T(P2)=4, T(P3)=16; Wartezeiten W = T - Burst: W(P1)=6, W(P2)=2, W(P3)=8
- Durchschnittliche Wartezeit (6+2+8)/3 = 5,33

Beurteilung: Round Robin reduziert die Reaktionszeit und gleicht CPU-Zeit besser aus, kann aber bei zu kleinem q mehr Kontextwechsel verursachen; FCFS ist einfach, aber anfällig für long Jobs und lange Wartezeiten anderer Prozesse.

(6) Zusammenfassung.
- Prozesse kapseln Programme mit eigenem Kontext; Threads ermöglichen leichten Parallelismus innerhalb eines Prozesses.
- Scheduling-Algorithmen beeinflussen Reaktionszeit, Durchsatz und Fairness; Preemption ermöglicht bessere Interaktivität, erhöht aber Kontextwechsel-Overhead.
- Kontextwechsel ist der zentrale Mechanismus für die Umsetzung von Multitasking; Optimierungen zielen auf geringeren Overhead und gerechtere Verteilung der CPU-Zeit.