Betriebssysteme

Kap. 2: Prozesse und Threads
2.3 Scheduling

Stand: WS 10/11 (25.01.11)

Prof. Dr. Wolfgang Küchlin

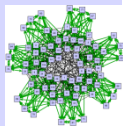
Dipl.-Inform., Dr. sc. techn. (ETH)

Arbeitsbereich Symbolisches Rechnen
Wilhelm-Schickard-Institut für Informatik
Fakultät für Informations- und Kognitionswissenschaften

Universität Tübingen

Steinbeis Transferzentrum
Objekt- und Internet-Technologien (OIT)

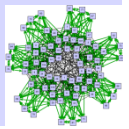
Wolfgang.Kuechlin@uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de

SR


Kap 2.3 Prozess-Ablaufplanung (Scheduling)

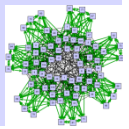
Inhalt

- Ziele einer Prozess-Ablaufplanung
- Karussell Laufplanung (round robin scheduling)
- Laufplanung mit Prioritäten (priority scheduling)
- Mehrfach-Warteschlangen
- Kürzester Auftrag zuerst
- 2 Ebenen Ablaufplanung



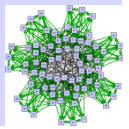
Ziele des Scheduling

- Fairness für die Prozesse
- Maximale HW-Auslastung
- Minimaler Scheduling-Aufwand
- Tolerierbare Antwortzeiten
- Bevorzugung von Prozessen mit erwünschtem Verhalten
- Sinnvolles Verhalten bei Überlast
- Behandeln von Prozessen gemäß ihrer Wichtigkeit
- Konsistentes und voraussagbares Schedulingverhalten

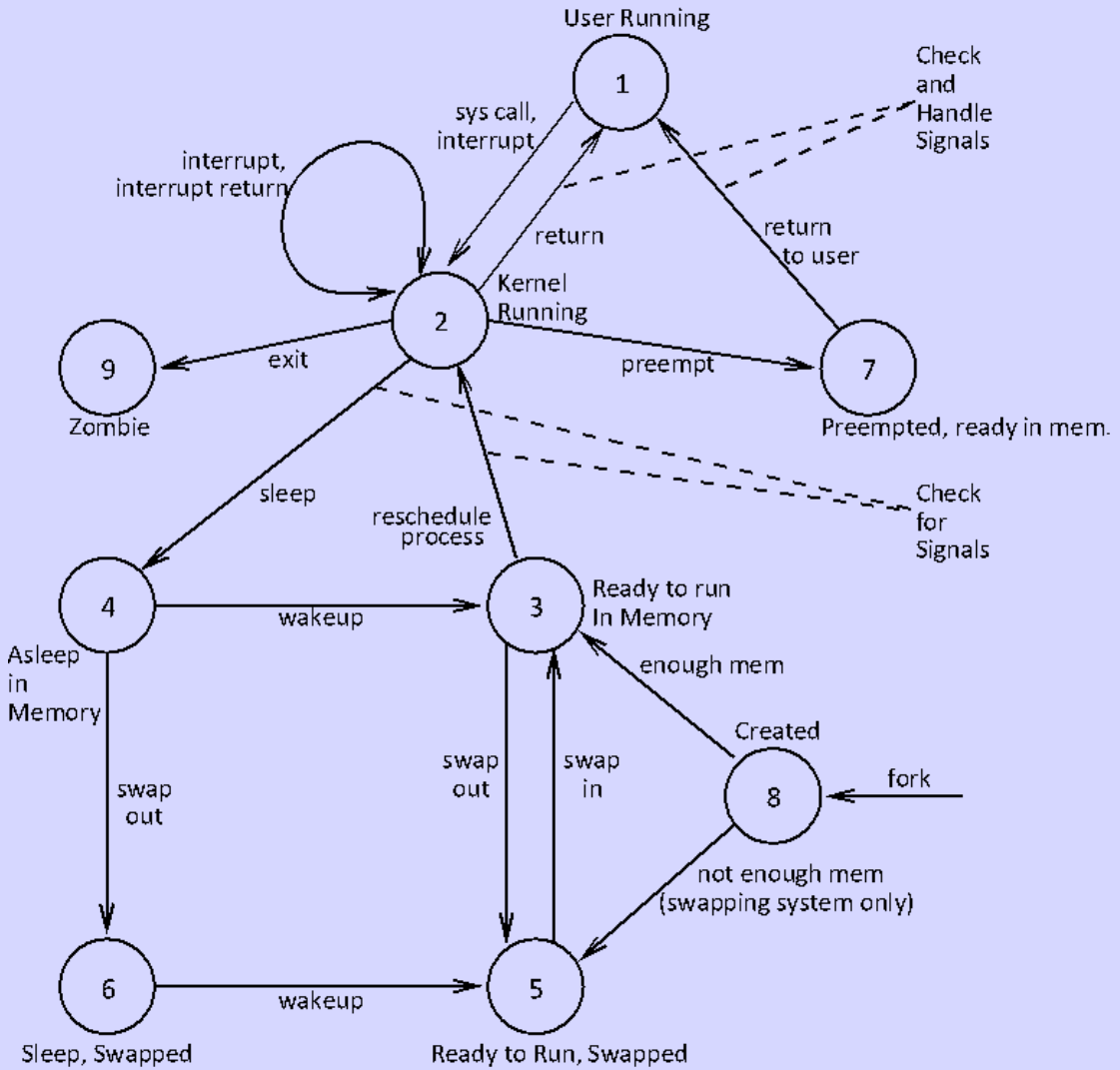


Scheduling

- Multiprogramming
  - CPU Auslastung maximieren
- Time-Sharing
  - Wechsel der Prozesse so schnell, dass interaktives Arbeiten möglich ist.
- Realzeit-Systeme
  - Reaktionszeiten garantieren (Echtzeit → Rechtzeitig)
- Queues
  - Job Queue
  - Ready Queue
  - Device Queue
- Prozesse migrieren zwischen den Queues.

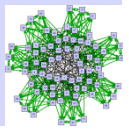


Prozess-Zustände in UNIX



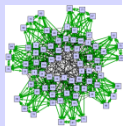
Scheduling-Warteschlange
Lineare Liste von Prozessen (PCBs)

- Kurzfristiger Scheduler
  - verwaltet CPU
  - wählt Prozesse aus und teilt ihnen CPU zu (dispatching)
- Mittelfristiger Scheduler (Swapper)
  - entscheidet, welche Prozesse im Hauptspeicher bleiben und welche ausgelagert werden. Unterstützt Speicherverwaltung und gleicht Systemlast aus.
- Langfristiger Scheduler (Job-Scheduler in Batch-Systemen)
  - entscheidet, welche Programme als nächstes gestartet (Prozesse zum System zulassen und in Hauptspeicher laden)



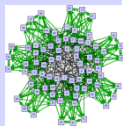
Scheduling-Warteschlange
Lineare Liste von Prozessen (PCBs)

- Dispatcher
  - übernimmt Aufsetzen des Prozesses nach Auswahl durch Scheduler
    - Context Switching/Umschaltung in User Mode / Übergeben der Kontrolle an den Prozess



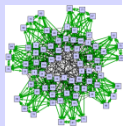
Non-Preemptive Scheduling

- Non-preemptive: Kein Entzug des Prozessors
  - feste Priorität (keine Anpassung nach Ressourcenverbrauch)
  - harte Form: nur freiwillige Aufgabe: *yield* (cooperative sched.)
  - weiche Form: Entzug nur durch Prozess höherer Priorität
- Laufender Prozess bleibt auf CPU bis er entweder
  - blockiert oder terminiert
  - weiche Form: Prozess höherer Priorität lauffähig wird
- Max. Laufzeit eines Prozesses vorhersagbar (keine unfreiwillige Unterbrechung unbestimmter Dauer)
- Einsatz in Realzeitsystemen
  - Systeme mit deterministischem Verhalten: Vollständiges Ablaufdesign.



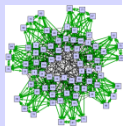
Preemptive Scheduling

- Preemption (=Verdrängung)
  - Allgemein in BS: Entzug von Betriebsmitteln
  - Beim Scheduling: Entzug des Prozessors
- Laufender Prozess wird vom Prozessor genommen bei
  - Termination
  - Blockierung nach system call (Wechsel *running* → *blocked*)
  - Preemption
    - für Scheduling relevantes Ereignis tritt ein
      - Interrupts, insbesondere Timer-Interrupts
      - System call
    - Neuer Prozess höherer Priorität wird lauffähig
    - Priorität des laufenden Prozesses ändert sich (z.B. durch Ressourcenverbrauch)
    - Scheduler bestimmt Prioritäten neu
    - Scheduler findet anderen Prozess höchster Priorität
    - Scheduler verdrängt bisherigen Prozess (*running* → *ready*)
    - Scheduler veranlasst dispatching des neuen Prozesses höchster Priorität



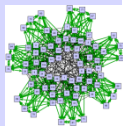
Einsatz des Preemptive Scheduling

- Priorität von wichtigen Prozessen im Bedarfsfall
  - Bei reaktiven Systemen (Realzeit) u.U. sofortige Reaktion auf externe Ereignisse
    - > laufender Prozeß wird verdrängt.
- Akzeptable Antwortzeiten (Multitasking)
  - Bei interaktiven Systemen (Timesharing) soll ein CPU-lastiger Prozeß (z.B. Simulationsberechnung) regelmäßig unterbrochen werden von interaktiven Prozessen
    - z.B. interaktive Dateneingabe durch Benutzer



Beeinflussung des Scheduling

- Folgende Prozesseigenschaften können (abhängig vom eingesetzten Verfahren) das Scheduling beeinflussen
  - Laufzeitverhalten
    - I/O-lastiger Prozeß blockiert nach kurzer Laufzeit
    - CPU-lastiger Prozeß wird meistens preempted
  - Wichtigkeit des Prozesses
  - Ressourcenbedarf des Prozesses
    - Bisher verbrauchte Prozessorzeit
    - Bisherige Verweildauer im System
    - Speicherverhalten, Page Fault Rate
    - Bedarf an Ressourcen (neben Speicher/CPU)
    - Noch nötige Prozessorzeit bis zur Termination

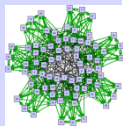


FIFO-Scheduling

- Einfache First-In First-Out Queue
- Neue Prozesse werden an die Queue angefügt
- Non-Preemptive
  - Prozess läuft bis zur Termination oder bis zum Blockieren
- Große durchschnittliche Antwortzeit

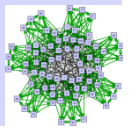
  —> ungeeignet für interaktive Systeme

Ungebräuchlich, enthalten in anderen Scheduling-Algorithmen



Round Robin Scheduling

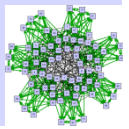
- FIFO-Scheduling mit Time-Slice
  - Preemptive
  - Limitierte Antwortzeiten
    - > geeignet für interaktive Systeme
- Bestimmung der Time-Slice
  - kurze Zeitscheibe ==> relativ großer Overhead durch Context Switch
    - kurze Antwortzeiten
  - lange Zeitscheibe ==> kleiner Overhead, lange Antwortzeiten
    - typischer Bereich: 10-100 ms



Round Robin Scheduling

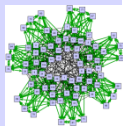
Timeslice/Zeitscheibe/Zeitquantum

Prozessor hat spezielles Register, das beim Laden des Prozesses entsprechend Zeitscheibe gesetzt wird.
Bei jedem Ablauf der HW-Zeitscheibe wird Register dekrementiert (parallel zur Bearbeitung des Prozesses), bei 0 wird HW-Interrupt ausgelöst.



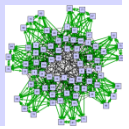
Priority Scheduling

- Unterteilung der Prozesse in Prioritätsklassen
- Separate Round Robin Queue für jede Priorität
- Höhere Priorität hat absoluten Bearbeitungs-Vorrang vor allen anderen.
  - Problem
    - > Prozesse mit tieferer Priorität können verhungern
  - Lösungen
    - (gewisse) Zeit für tiefere Prioritäten reservieren
    - Periodische Erhöhung der Prioritäten von nicht bearbeiteten Prozessen (=> Multi-Level Feedback Queue Scheduling)

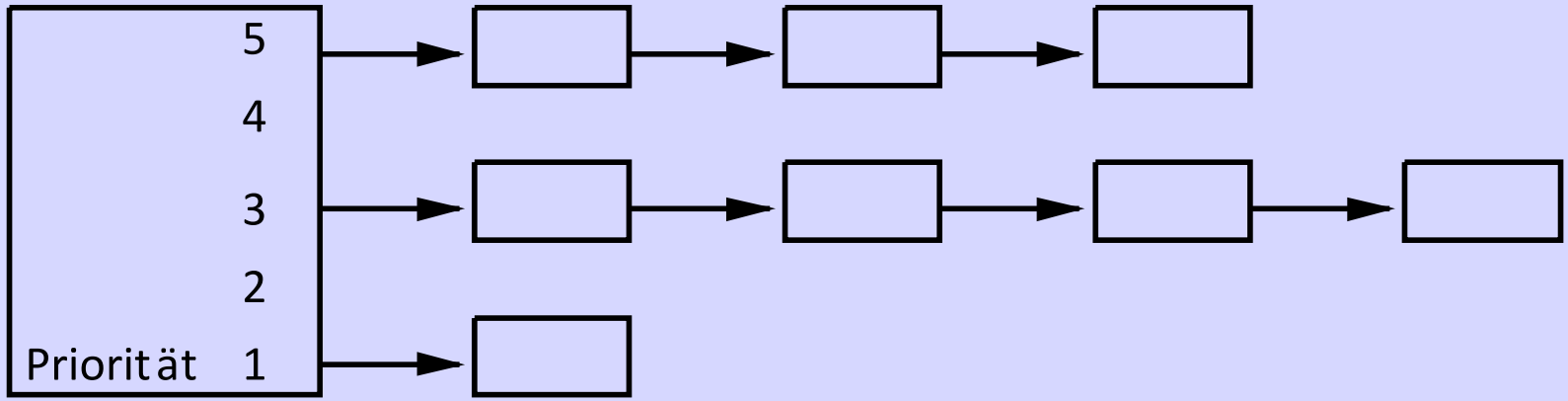


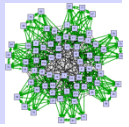
Multi-Level Feedback Queue Scheduling

- Zusätzlich
  - Veränderung der Prioritäten aufgrund des Prozeßverhaltens
    - blockiert: Priorität wird erhöht
    - preempted: Priorität wird gesenkt



Mehrfach-Warteschlangen

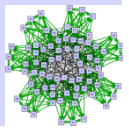




VMS Scheduling

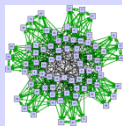
- Ziel
  - Gleichzeitige Bearbeitung von Realtime- und Timesharing-Applikationen
- Neue Scheduling Klassen-Einteilung
  - Realtime
  - System
  - User
- Preemptive

- Prioritätsgesteuerte Multilevel (32) Queue
  - Immmer der lauffähigste Prozeß mit der höchsten Proirität wird ausgewählt



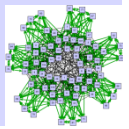
VMS Scheduling

- 0-15: Timesharing
  - Round Robin mit Timeslice innerhalb Prioritätsklasse
    - Prozessverhalten verändert Priorität (Feedback, Priorität verbleibt innerhalb 0-15)
    - Unterbrechung bei Ende der Timeslice oder falls Prozess mit höherer Priorität lauffähig wird
- 16-31: Realtime
  - FIFO innerhalb Prioritätsklasse
  - Feste Prioritäten
  - Unterbrechung nur falls Prozess mit höherer Priorität lauffähig wird.



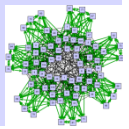
Scheduling in UNIX (BSD 4.4)

- Ziel
  - Timesharing, Bevorzugung von interaktiven Prozessen, nicht-interaktive Prozesse sollen aber nicht verhungern.
- Zuteilung von CPU-Zeit aufgrund der Prioritäten
- Multilevel Feedback
  - Mehrere Prioritätsebenen
  - Prioritätsanpassung nach feed-back (Ressourcennutzung)
- Preemption
  - nach Ablauf der Zeitscheibe (100ms)
  - wenn ein Prozess höherer Priorität lauffähig wird
    - Prozess im user mode wird sofort verdrängt
    - Prozess im Kernel Mode wird erst bei return-to-user-mode verdrängt (UNIX ist nicht realzeit-fähig)



Scheduling in UNIX (BSD 4.4)

- Feedback
  - Prozessornutzung geht in Priorität ein: wenn der Prozess läuft, wird sie gesenkt wenn er nicht läuft, wird sie erhöht.
  - Neuberechnung alle 4 clock ticks (4*10ms)
  - numerisch größere Werte → niedrigere Priorität
  - p_userpri = PUSER + [p_estcpu / 4] + 2*p_nice
    - PUSER = 50 (baseline prio for user processes)
    - -20 <= p_nice <= 20 (user settable)
    - p_estcpu zählt die clock-ticks, in denen der Prozess läuft
  - Einmal pro Sekunde wird p_estcpu gefiltert, um alte Werte zu vergessen
    - p_estcpu = [(2*load) / (2*load + 1)]*p_estcpu+p_nice
    - Die Systemlast *load* wird aus der durchschnittl. Länge der Run-Queues in der letzten Minute bestimmt



Scheduling in UNIX (BSD 4.4)

- Beispiel des p_estcpu Filters
  - p_estcpu = [(2*load) / 2*load + 1)]*p_estcpu+p_nice
  - Annahme: 1 Prozess im System
    - 1 Filterschritt: p_estcpu = [2/3]*p_estcpu + p_nice
  - Annahme: 1 Prozess, p_nice=0, p_estcpu = Ti in Intervall i
    - 1. Schritt: p_estcpu = [2/3]*T0
    - 2. Schritt: p_estcpu = [2/3]*(T1 + [2/3]*T0) = 0.66*T1+0.44*T0
    - 3. Schritt: p_estcpu = 0.66*T2 + 0.44*T1 + 0.30*T0
    - 4. Schritt: p_estcpu = 0.66*T3 + 0.44*T2 + 0.30*T1+0.20*T0
    - 5. Schritt: p_estcpu = 0.66*T4 + 0.44*T3 + 0.30*T2+0.20*T1+0.13*T0
  - Nach 5 Schritten gehen nur noch 13% des Verbrauchs T0 ein
  - Nach 5 Sekunden sind grob 90% des Verbrauchs vergessen.
- Nach sleep von *n* sec: $p\_estcpu = p\_estcpu * [2*load / (2*load + 1)]^n$