

# Betriebssysteme (BS)

## Fadenverwaltung

Daniel Lohmann

Lehrstuhl für Informatik 4  
Verteilte Systeme und Betriebssysteme

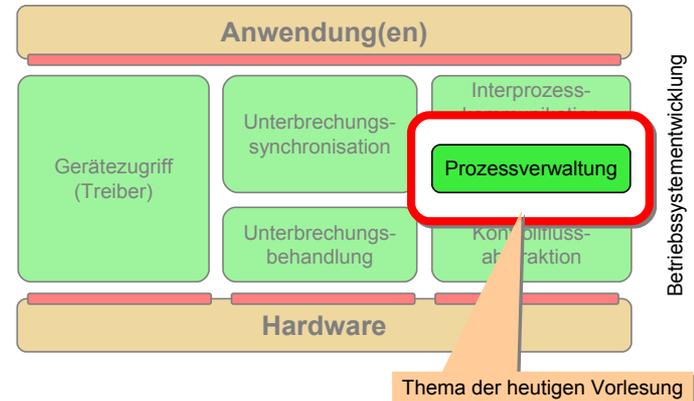


## Agenda

- Betriebssystemfäden
  - Motivation
  - kooperativer Fadenwechsel
  - präemptiver Fadenwechsel
  - Arbeitsteilung
- Ablaufplanung
  - Grundbegriffe und Klassifizierung
  - unter Linux
  - unter Windows (XP/2000/2003)
- Zusammenfassung



## Überblick: Vorlesungen



## Agenda

- **Betriebssystemfäden**
  - **Motivation**
  - **kooperativer Fadenwechsel**
  - **präemptiver Fadenwechsel**
  - **Arbeitsteilung**
- Ablaufplanung
  - Grundbegriffe und Klassifizierung
  - unter Linux
  - unter Windows (XP/2000/2003)
- Zusammenfassung

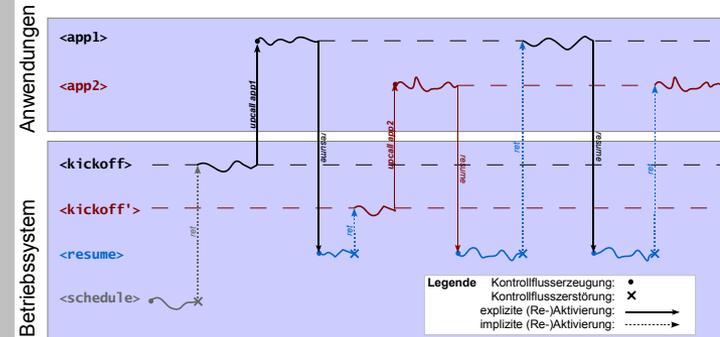


## Betriebssystemfäden: Motivation

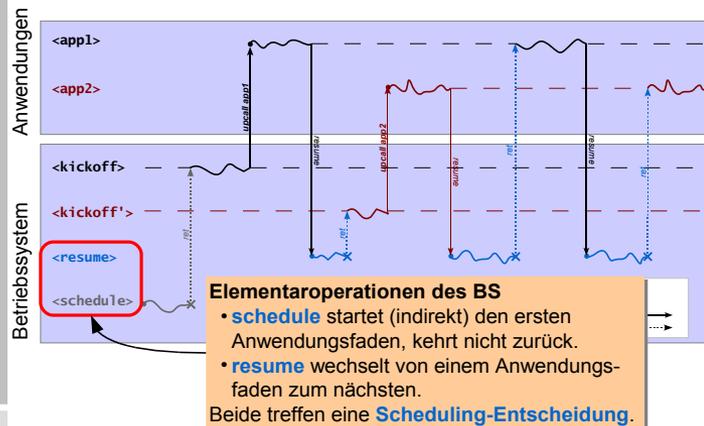
- Ansatz: Anwendungen „unbemerkt“ als eigenständige Fäden ausführen
  - eine BS-Koroutine pro Anwendung
  - Aktivierung der Anwendung erfolgt durch Aufruf
  - Koroutinenwechsel erfolgt indirekt durch Systemaufruf
- Vorteile
  - unabhängige Anwendungsentwicklung
  - Ablaufplanung (*Scheduling*) wird zentral implementiert
  - bei E/A kann eine Anwendung einfach vom BS „blockiert“ und später wieder „geweckt“ werden
  - zusätzlicher Entzugsmechanismus (*preemption mechanism*) kann die Monopolisierung der CPU verhindern



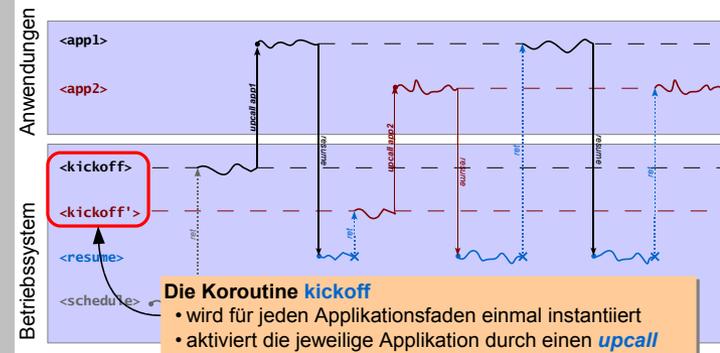
## Kooperativer Fadenwechsel



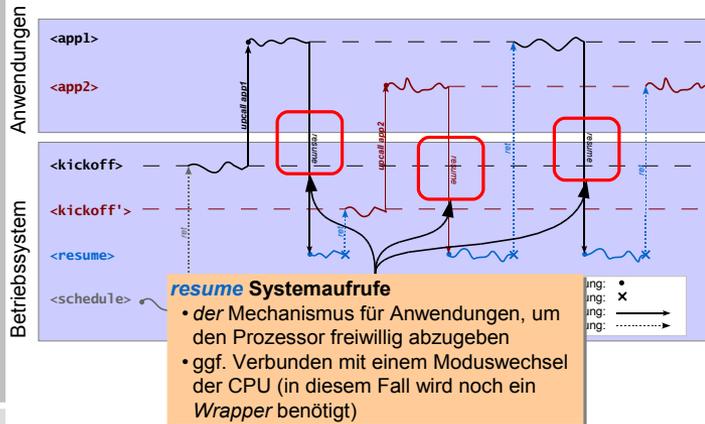
## Kooperativer Fadenwechsel



## Kooperativer Fadenwechsel

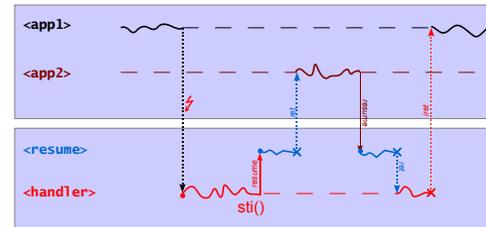


## Kooperativer Fadenwechsel



## Präemptiver Fadenwechsel

- CPU-Entzug durch Zeitgeberunterbrechung
  - die Unterbrechung ist „nur“ ein impliziter Aufruf
  - Behandlungsroutine kann *resume* aufrufen



## Arbeitsteilung

- **Scheduler**
  - trifft **strategische Entscheidungen** zur Ablaufplanung
  - betrachtet wird immer eine Menge lauffähiger Fäden
    - die Fäden sind allgemein in einer CPU-Warteschlange aufgereiht
    - die Sortierung erfolgt entsprechend der **Scheduling-Strategie**
  - laufende Faden ist immer von der Entscheidung mit betroffen
    - dazu muss der laufende Faden jederzeit „greifbar“ sein
    - vor der Umschaltung ist der laufende Faden zu vermerken
  - ein ausgewählter neuer Faden wird dem *Dispatcher* übergeben
- **Dispatcher**
  - setzt die Entscheidungen durch und schaltet Fäden (mit Hilfe von *resume*) um

## Agenda

- Betriebssystemfäden
  - Motivation
  - kooperativer Fadenwechsel
  - präemptiver Fadenwechsel
  - Arbeitsteilung
- **Ablaufplanung**
  - **Grundbegriffe und Klassifizierung**
    - unter Linux
    - unter Windows (XP/2000/2003)
- Zusammenfassung

## Ablaufplanung: Einteilung ...

- nach der **Betriebsmittelart**  
der zeitweilig belegten Hardware-Ressourcen
- nach der **Betriebsart**  
des zu bedienenden/steuernden Rechnersystems
- nach dem **Zeitpunkt**  
der Erstellung des Ablaufplans
- nach der **Vorhersagbarkeit**  
von Zeitpunkt und Dauer von Prozessabläufen
- nach dem **Kooperationsverhalten**  
der (Benutzer/System-) Programme
- nach der **Rechnerarchitektur**  
des Systems
- nach der **Ebene der Entscheidungsfindung**  
bei der Betriebsmittelvergabe



## ... nach der Betriebsmittelart

- **CPU scheduling**  
des Betriebsmittels "CPU"
  - die Prozessanzahl zu einem Zeitpunkt ist höher als die Prozessoranzahl
  - ein Prozessor ist zwischen mehreren Prozessen zu multiplexen
  - Prozesse werden dem Prozessor über eine Warteschlange zugeteilt
- **I/O scheduling**  
des Betriebsmittels "Gerät", speziell: "Platte"
  - gerätespezifische Einplanung der von Prozessen abgesetzten E/A-Aufträge
  - *disk scheduling*, z.B., berücksichtigt typischerweise drei Faktoren:
    - (1) Positionszeit, (2) Rotationszeit, (3) Transferzeit
  - Geräteparameter und Gerätezustand bestimmen die nächste E/A-Aktion
  - die getroffenen Entscheidungen sind ggf. nicht konform zum *CPU scheduling*



## ... nach der Betriebsart

- **batch scheduling**  
interaktionsloser bzw. unabhängiger Programme
  - nicht-verdrängende bzw. verdrängende Verfahren mit langen Zeitscheiben
  - Minimierung der Kontextwechsellanzahl
- **interactive scheduling**  
interaktionsreicher bzw. abhängiger Programme
  - ereignisgesteuerte, verdrängende Verfahren mit kurzen Zeitscheiben
  - Antwortzeitminimierung durch Optimierung der Systemaufrufe
- **real-time scheduling**  
zeitkritischer bzw. abhängiger Programme
  - ereignis- oder zeitgesteuerte **deterministische** Verfahren
  - Garantie der Einhaltung umgebungsbedingter Zeitvorgaben
  - Rechtzeitigkeit ist entscheidend und nicht Geschwindigkeit



## ... nach dem Zeitpunkt

- **online scheduling**  
dynamisch, **während** der eigentlichen Ausführung
  - interaktive- und Stapelsysteme, aber auch weiche Echtzeitsysteme
- **offline scheduling**  
statisch, **vor** der eigentlichen Ausführung
  - wenn die Komplexität eine Ablaufplanung im laufenden Betrieb verbietet
    - Einhaltung aller Zeitvorgaben garantieren: ein NP-vollständiges Problem
    - kritisch, wenn auf jede abfangbare katastrophale Situation zu reagieren ist
  - Ergebnis der Vorberechnung ist ein vollständiger Ablaufplan (in Tabellenform)
    - (semi-) automatisch erstellt per Quelltextanalyse spezieller "Übersetzer"
    - oft zeitgesteuert abgearbeitet/ausgeführt als Teil der Prozessabfertigung
  - die Verfahren sind zumeist beschränkt auf strikte Echtzeitsysteme



## ... nach der Vorhersagbarkeit

- **deterministic scheduling**  
bekannter, exakt vorberechneter Prozesse
  - Prozesslaufzeiten/-termine sind bekannt, sie wurden ggf. "offline" berechnet
  - die genaue Vorhersage der CPU-Auslastung ist möglich
  - das System garantiert die Einhaltung der Prozesslaufzeiten/-termine
  - die Zeitgarantien gelten unabhängig von der jeweiligen Systemlast
- **probabilistic scheduling**  
unbekannter Prozesse
  - Prozesslaufzeiten/-termine bleiben unbestimmt
  - die (wahrscheinliche) CPU-Auslastung kann lediglich abgeschätzt werden
  - das System kann Zeitgarantien nicht geben und auch nicht einhalten
  - Zeitgarantien sind durch Anwendungsmaßnahmen bedingt erreichbar



## ... nach dem Kooperationsverhalten

- **cooperative scheduling**  
von einander abhängiger Prozesse
  - Prozesse müssen die CPU freiwillig abgeben, zugunsten anderer Prozesse
  - die Programmausführung muss (direkt/indirekt) **Systemaufrufe** bewirken
  - die Systemaufrufe müssen (direkt/indirekt) den *Scheduler* aktivieren
- **preemptive scheduling**  
von einander unabhängiger Prozesse
  - Prozessen wird die CPU entzogen, zugunsten anderer Prozesse
  - **Ereignisse** können die Verdrängung des laufenden Prozesses bewirken
  - die Ereignisverarbeitung aktiviert (direkt/indirekt) den *Scheduler*



## ... nach der Rechnerarchitektur

- **uni-processor scheduling**  
in Mehr{programm,prozess}systemen
  - die Verarbeitung von Prozessen kann nur pseudo-parallel erfolgen
- **multi-processor scheduling**  
in Systemen mit gemeinsamen Speicher
  - jeder Prozessor arbeitet seine lokale Warteschlange ab:
    - die Prozesse sind den Prozessoren (i.d.R.) fest zugeordnet
    - Prozessoren können leer laufen, obwohl noch Prozesse ausführbar sind
  - alle Prozessoren arbeiten eine globale Warteschlange ab:
    - die Prozesse sind den Prozessoren nicht fest zugeordnet
    - Prozessoren laufen erst leer, wenn keine Prozesse mehr ausführbar sind
  - die parallele Verarbeitung von Prozessen wird ermöglicht



## ... nach der Ebene

- **long-term scheduling** [s – min]  
kontrolliert den Grad an Mehrprogrammbetrieb
  - Benutzer Systemzugang gewähren, Programme zur Ausführung zulassen
  - Prozesse dem *medium-* bzw. *short-term scheduling* zuführen
- **medium-term scheduling** [ms – s]  
als Teil der Ein-/Auslagerungsfunktion
  - Programme zwischen Vorder- und Hintergrundspeicher hin- und herbewegen
  - *swapping*: auslagern (*swap-out*), einlagern (*swap-in*)
- **short-term scheduling** [µs – ms]  
regelt die Prozessorzuteilung an die Prozesse
  - ereignisgesteuerte Ablaufplanung: Unterbrechungen, Systemaufrufe, Signale
  - Blockierung bzw. Verdrängung des laufenden Prozesses



## Scheduling-Kriterien

- **Antwortzeit** Minimierung der Zeitdauer von der Auslösung einer Systemanforderung bis zur Entgegennahme der Rückantwort, bei gleichzeitiger Maximierung der Anzahl interaktiver Prozesse.
- **Durchlaufzeit** Minimierung der Zeitdauer vom Starten eines Prozesses bis zu seiner Beendigung, d.h., der effektiven Prozesslaufzeit und aller Prozesswartezeiten.
- **Termineinhaltung** Starten und/oder Beendigung eines Prozesses zu einem fest vorgegebenen Zeitpunkt.
- **Vorhersagbarkeit** Deterministische Ausführung des Prozesses unabhängig von der jeweils vorliegenden Systemlast.
- **Durchsatz** Maximierung der Anzahl vollendeter Prozesse pro vorgegebener Zeiteinheit. Liefert ein Maß für die geleistete Arbeit im System.
- **Prozessorauslastung** Maximierung des Prozentanteils der Zeit, während der die CPU Prozesse ausführt, d.h., "sinnvolle" Arbeit leistet.
- **Gerechtigkeit** Gleichbehandlung der auszuführenden Prozesse und Zusicherung, den Prozessen innerhalb gewisser Zeiträume die CPU zuzuteilen.
- **Dringlichkeiten** Bevorzugte Verarbeitung des Prozesses mit der höchsten (statisch/dynamisch zugeordneten) Priorität.
- **Lastausgleich** Gleichmäßige Betriebsmittelauslastung bzw. bevorzugte Verarbeitung der Prozesse, die stark belastete Betriebsmittel eher selten belegen.



## Scheduling-Kriterien

- **Antwortzeit**
- **Durchlaufzeit**
- **Termineinhaltung**
- **Vorhersagbarkeit**
- **Durchsatz**
- **Prozessorauslastung**
- **Gerechtigkeit**
- **Dringlichkeiten**
- **Lastausgleich**

### Benutzerorientierte Kriterien

- wahrgenommenes Systemverhalten
- bestimmen die Akzeptanz durch Benutzer



beeinflussen

### Systemorientierte Kriterien

- effiziente Nutzung der Betriebsmittel
- bestimmen die Kosten des Rechnerbetriebs



## Kriterien bei typischen Betriebsarten

- allgemein (unabhängig von der Betriebsart)
  - Gerechtigkeit
  - Lastausgleich
- **Stapelsysteme**
  - Durchsatz
  - Durchlaufzeit
  - Prozessorauslastung
- **interaktive Systeme**
  - Antwortzeit (Proportionalität – Bearbeitungsdauer entspricht Erwartung)
- **Echtzeitsysteme**
  - Dringlichkeit
  - Termineinhaltung
  - Vorhersagbarkeit



## Agenda

- Betriebssystemfäden
  - Motivation
  - kooperativer Fadenwechsel
  - präemptiver Fadenwechsel
  - Arbeitsteilung
- Ablaufplanung
  - Grundbegriffe und Klassifizierung
  - **unter Linux**
  - unter Windows (XP/2000/2003)
- Zusammenfassung

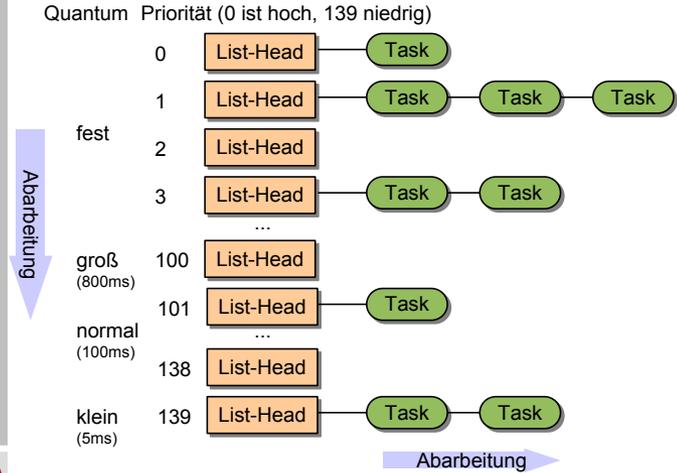


## Linux Tasks ...

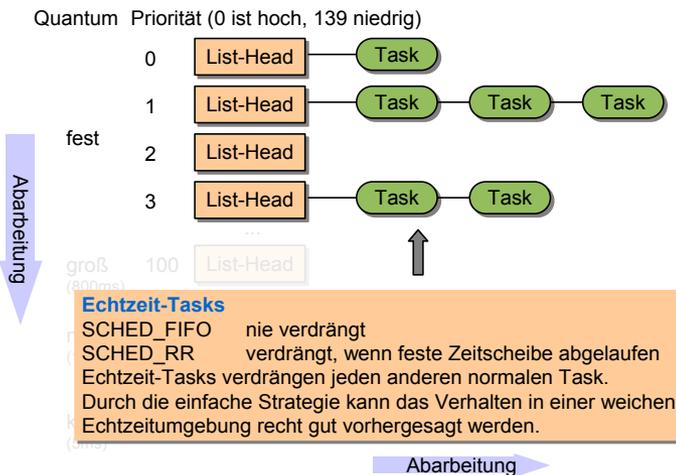
- sind die **Linux Kernel Abstraktion** für ...
  - UNIX Prozesse:** ein Kontrollfaden in einem Adressraum
  - Linux Threads:** spezieller Prozess, der sich seinen virtuellen Adressraum mit mindestens einem anderen *Thread* teilt
- sind die vom Scheduler betrachteten Aktivitäten
  - ein Programm mit vielen Threads bekommt unter Linux mehr Rechenzeit als ein klassischer Prozess
  - gleiches gilt allerdings auch für ein Programm mit einem Prozess und vielen Kindprozessen



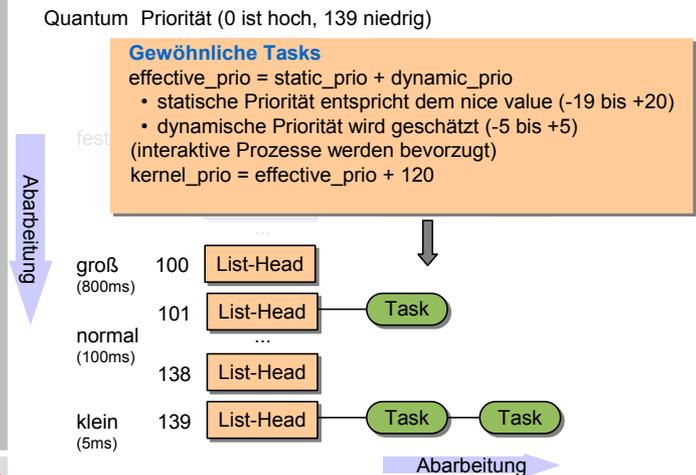
## Multi-Level Queues



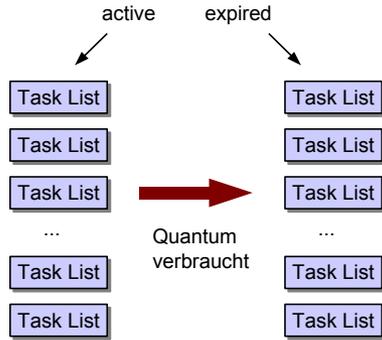
## Multi-Level Queues



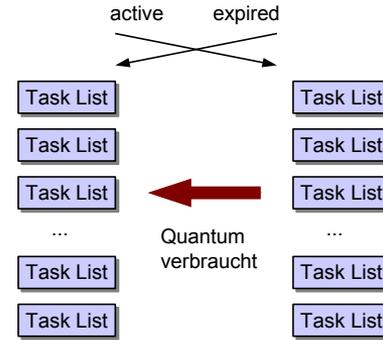
## Multi-Level Queues



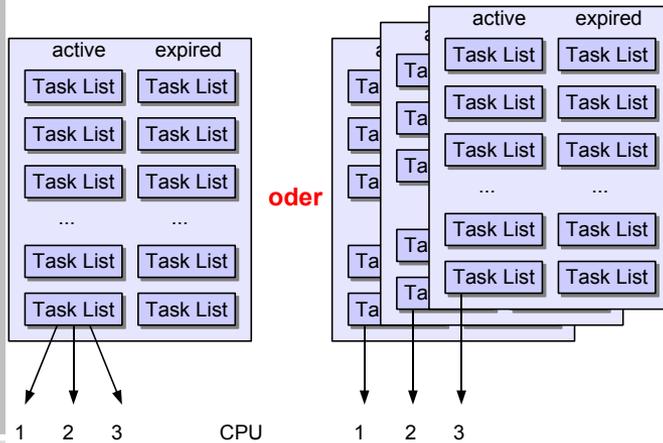
## Active und Expired



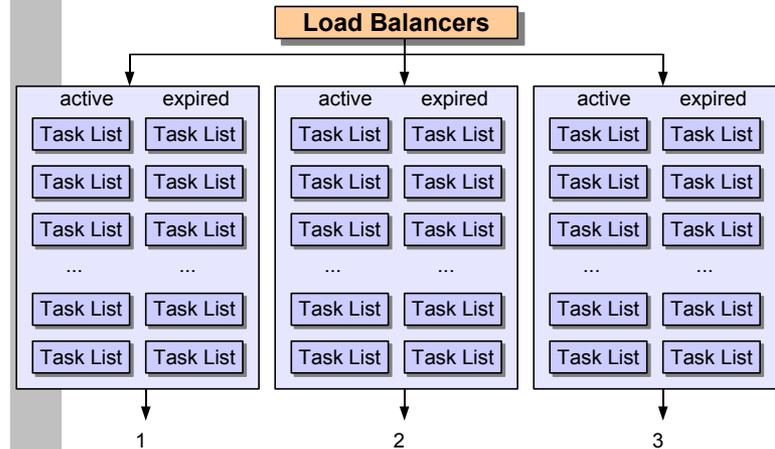
## Active und Expired



## Bereit-Listen im SMP Betrieb



## Lastausgleich



## Fazit Linux

- „*interactive, probabilistic, online, preemptive, multi-processor CPU scheduling*“
- Bevorzugung interaktiver Prozesse
  - schnelle Reaktion auf Eingaben
  - gleichzeitig Fortschrittgarantie für CPU-lastige Prozesse
- $O(1)$  bei allen Operationen des Scheduler
  - Einfügen, Entfernen, Scheduling-Entscheidung
- Mehrprozessorunterstützung
  - Mehrere Bereit-Listen: Parallele Scheduler Ausführung
  - Keine Idle-Phasen (war ein Problem beim alten Linux Scheduler)
  - CPU-Lastausgleich



## Agenda

- Betriebssystemfäden
  - Motivation
  - kooperativer Fadenwechsel
  - präemptiver Fadenwechsel
  - Arbeitsteilung
- Ablaufplanung
  - Grundbegriffe und Klassifizierung
  - unter Linux
  - **unter Windows (XP/2000/2003)**
- Zusammenfassung

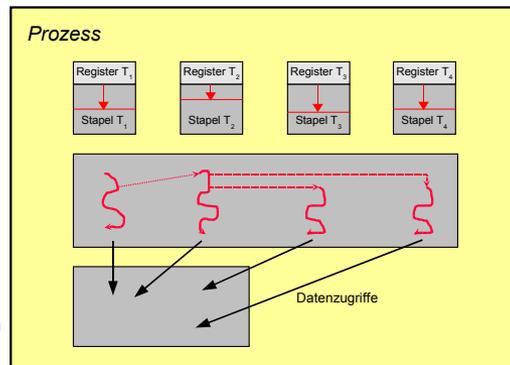


## Prozesse und Fäden in Win NT

Stapel +  
Registersatz  
(1 je Faden)

Code

Gobale und  
statische Daten



## Prozesse und Fäden in Win NT

- Prozess: Umgebung und Adressraum für Fäden
  - Ein Win32 Prozess enthält immer mindestens einen Faden
  - Faden (engl. thread): Code ausführende Einheit
- Fadenimplementierung wird durch den NT Systemkern erbracht
  - *Usermode-Threads* möglich („*Fibers*“), aber unüblich
- „*Threads*“ bekommen vom *Scheduler* Rechenzeit zugeteilt



## Der NT-Scheduler

- Preemptives, prioritätengesteuertes Scheduling:
  - Thread mit höherer Priorität verdrängt Thread niedrigerer Priorität
    - Egal ob Thread sich im User- oder Kernelmode befindet
    - Die meisten Funktionen der Executive („Kernel“) sind ebenfalls als Threads implementiert
  - Round-Robin bei Threads gleicher Priorität
    - Zuteilung erfolgt reihum für eine Zeitscheibe (Quantum)
- Thread-Prioritäten
  - Derzeit 0 bis 31, aufgeteilt in drei Bereiche
    - Variable Priorities: 1 bis 15
    - Realtime Priorities: 16 bis 31
    - Priorität 0 ist reserviert für den Nullseiten-Thread
  - Threads der Executive verwenden maximal Priorität 23



## Zeitscheiben (Quantum)

	Kurze Quantumwerte		Lange Quantumwerte	
	Variabel	Fix	Variabel	Fix
Thread in HG-Prozess	6	18	12	36
Thread in VG-Prozess	12	18	24	36
Aktiver Thread in VG-Prozess	18	18	36	36

- Quantum wird vermindert
  - um den Wert 3 bei jedem Clock-Tick (alle 10 bzw. 15 msec)
  - um den Wert 1, falls Thread in den Wartezustand geht
- Länge einer Zeitscheibe: 20 – 180 msec



## Prioritätsklassen, relative Threadpriorität

Relative Thread Priority	Process Priority Class					
	Idle	Below Normal	Normal	Above Normal	High	Realtime
Time Critical =15	15	15	15	15	15	31
Highest +2	6	8	10	12	15	26
Above Normal +1	5	7	9	11	14	25
Normal	4	6	8	10	13	24
Below Normal -1	3	5	7	9	12	23
Lowest -2	2	4	6	8	11	22
Idle =1	1	1	1	1	1	16



## Prioritäten: Variable Priorities

- Variable Priorities (1-15)
  - Scheduler verwendet Strategien, um „wichtige“ Threads zu bevorzugen
    - *Quantum-Stretching* (Bevorzugung des aktiven GUI-Threads)
    - dynamische Anhebung (*Boost*) der Priorität für wenige Zeitscheiben bei Ereignissen
  - Fortschrittsgarantie
    - Alle 3 bis 4 Sekunden bekommen bis zu 10 „benachteiligte“ Threads für zwei Zeitscheiben die Priorität 15
  - Threadpriorität berechnet sich wie folgt (vereinfacht):

**Prozessprioritätsklasse + Threadpriorität + Boost**



## Prioritäten: *Realtime Priorities*

- *Realtime Priorities* (16-31)
  - Reines prioritätengesteuertes Round-Robin
    - Keine Fortschrittsgarantie
    - Keine dynamische Anhebung
    - Betriebssystem kann negativ beeinflusst werden
    - Spezielles Benutzerrecht erforderlich (SeIncreaseBasePriorityPrivilege)
  - Threadpriorität berechnet sich wie folgt:
 

**REALTIME\_PRIORITY\_CLASS + Threadpriorität**

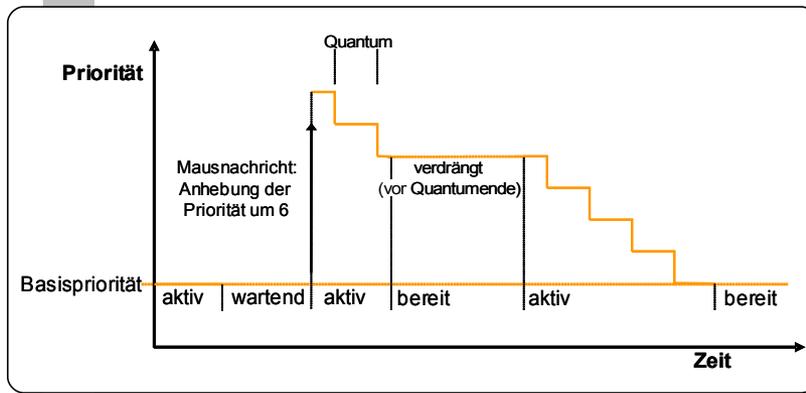


## Dynamische Prioritätsanpassung

- *Dynamic Boosts*
  - Thread-Prioritäten werden vom System in bestimmten Situationen dynamisch angehoben (nicht bei REALTIME\_PRIORITY\_CLASS)
    - Platten-Ein- oder Ausgabe abgeschlossen: +1
    - Maus, Tastatureingabe: +6
    - Semaphore, Event, Mutex: +1
    - Andere Ereignisse (Netzwerk, Pipe,...): +2
    - Ereignis in Vordergrundapplikation: +2
  - Dynamic Boost wird „verbraucht“ (eine Stufe pro Quantum)

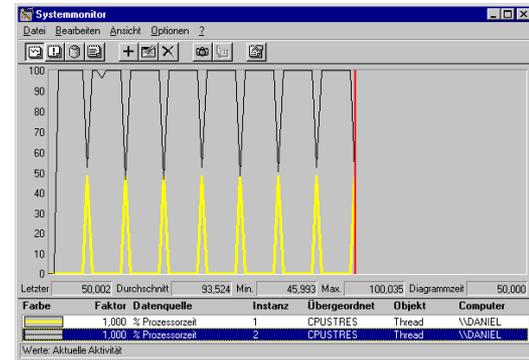


## Prioritätsänderung nach einem Boost



## Der *Balance-Set-Manager*

- Etwa alle 3-4 Sekunden erhalten bis zu 10 „benachteiligte“ Threads für zwei Zeitscheiben die Priorität 15
  - Implementierung der Fortschrittsgarantie



## Auswahl des nächsten Threads (SMP)

Ziel: „gerechtes“ RoundRobin bei max. Durchsatz

Problem: Cache-Effekte

Affinität (Zuordnung von CPUs zu Thread):

- **hard\_affinity:** Feste Zuordnung  
→ explizit durch `SetThreadAffinity()` zugewiesen
- **ideal\_processor:** „Ideale“ Zuordnung  
→ implizit bei Erzeugung zugewiesen („zufällig“)  
→ änderbar mit `SetThreadIdealProcessor()`
- **soft\_affinity:** Letzte CPU, auf welcher der Thread lief  
→ intern vom Scheduler verwaltet
- **last\_run:** Zeitpunkt der letzten Zuweisung zu einer CPU  
→ intern vom Scheduler verwaltet



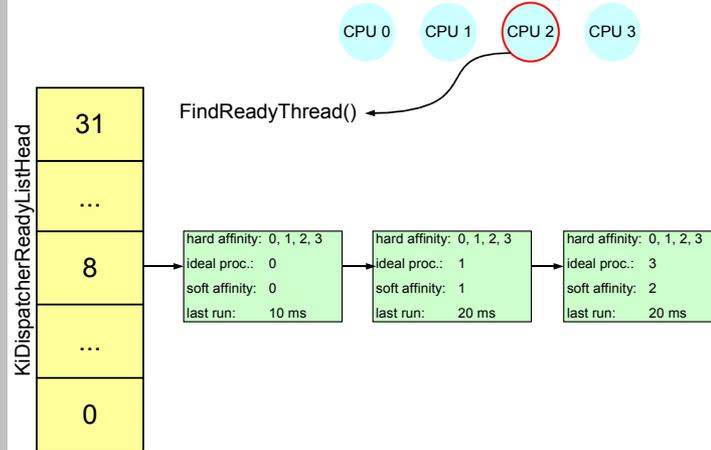
## Auswahl des nächsten Threads (SMP)

▪ Algorithmus: CPU  $n$  ruft `FindReadyThread()` auf

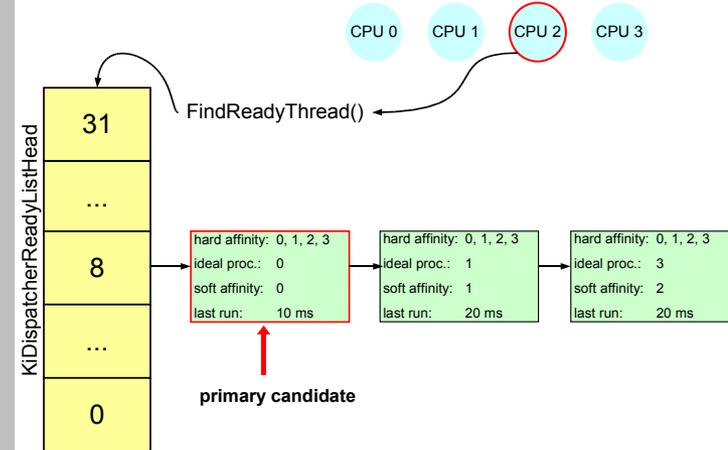
- Wähle höchstpriorie, nicht-leere Warteschlange
- Suche in dieser Warteschlange nach Thread, mit
  - `soft_affinity == n` oder
  - `ideal_processor == n` oder
  - `currentTime() - last_run > 2 Quantum` oder
  - `priority >= 24`
- Sonst wähle Kopf der Warteschlange



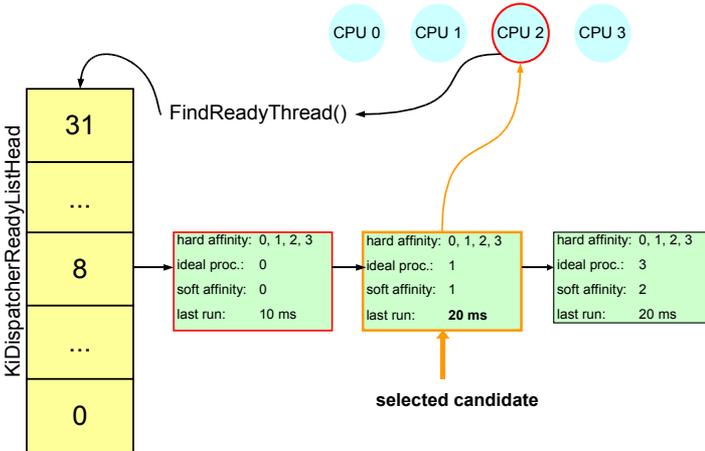
## Auswahl des nächsten Threads (SMP)



## Auswahl des nächsten Threads (SMP)



## Auswahl des nächsten Threads (SMP)



## Änderungen in Windows 2003

- Eine ReadyQueue pro CPU
- Algorithmus: CPU n ruft FindReadyThread() auf
  - Wähle höchstprior, nicht-leere Warteschlange von CPU n
  - Wähle Kopf dieser Warteschlange
- Falls ReadyQueue komplett leer ist, aktiviere Idle-Loop
- Im Idle-Loop: Durchsuche ReadyQueue anderer CPUs



## Fazit Windows NT

- „*interactive, probabilistic, online, preemptive, multi-processor CPU scheduling*“
- Prioritätenmodell erlaubt feine Zuteilung der Prozessorzeit
  - Dynamische Anpassungen beachten
  - Usermode-Threads mit hohen Echtzeitprioritäten haben Vorrang vor allen System-Threads!
  - Executive ist im allgemeinen unterbrechbar
- Interaktive Threads können bevorzugt werden
  - Insbesondere GUI/Multimedia-zentrierte Threads
- Weitere Verbesserungen für SMP in Windows 2003



## Agenda

- Betriebssystemfäden
  - Motivation
  - kooperativer Fadenwechsel
  - präemptiver Fadenwechsel
  - Arbeitsteilung
- Ablaufplanung
  - Grundbegriffe und Klassifizierung
  - unter Linux
  - unter Windows (XP/2000/2003)

■ **Zusammenfassung**



## Zusammenfassung

---

- *Threads* sind Koroutinen des Betriebssystems
  - BS hat Entzugsmechanismen
  - Strategie der Ablaufplanung wird als *Scheduling* bezeichnet
- *Scheduling* hat großen Einfluss auf die Performanz des Gesamtsystems, es legt fest, ...
  - welche Prozesse warten und welche voranschreiten
  - welche Betriebsmittel wie ausgelastet sind
- Es gibt verschiedenste Varianten des Scheduling
  - nur wenige Unterschiede bei gängigen PC/Workstation Betriebssystemen
  - eventuell aber starke Unterschiede in anderen Anwendungsdomänen

