

# Der Scheduler von Windows

## Konzepte und Strategien

Daniel Lohmann

lohmann@informatik.uni-erlangen.de

## Gliederung

1. Grundbegriffe
2. Eigenschaften des Schedulers
  - ▶ Grundlegende Eigenschaften
  - ▶ Prioritätenmodell
  - ▶ Dynamische Prioritätenanpassungen
3. Interner Aufbau
  - ▶ Interruptverarbeitung
  - ▶ Aufruf des Schedulers
4. Fazit

Der Scheduler von Windows – Konzepte und Strategien

Folie 2

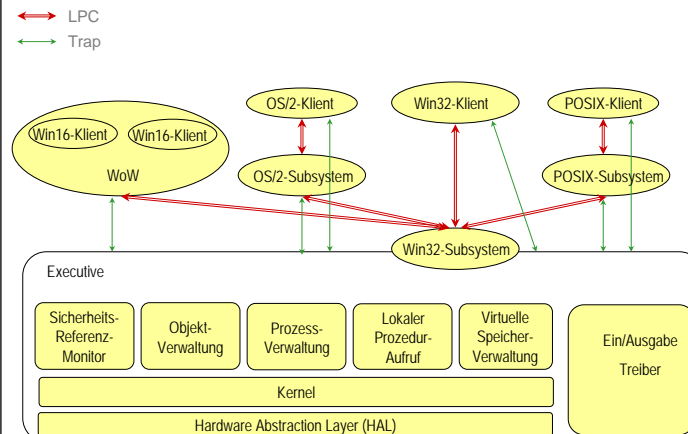
## (Einige) (ursprüngliche) Designziele von Windows NT

- ▶ Anwendungsportabilität
  - ▶ OS/2, Posix, Win16, Win32
- ▶ Plattformunabhängigkeit
  - ▶ Diverse RISC/CISC Architekturen
  - ▶ Ursprünglich MIPS, PowerPC, Alpha, I386
- ▶ Unterstützung für SMP

Der Scheduler von Windows – Konzepte und Strategien

Folie 3

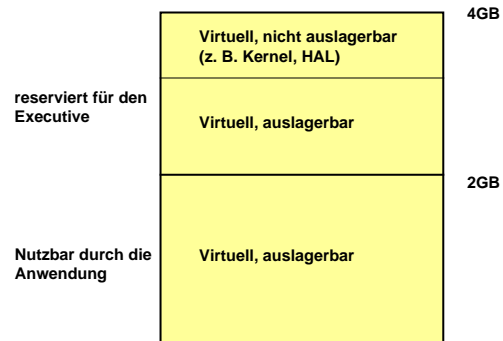
## Architektur von Windows NT



Der Scheduler von Windows – Konzepte und Strategien

Folie 4

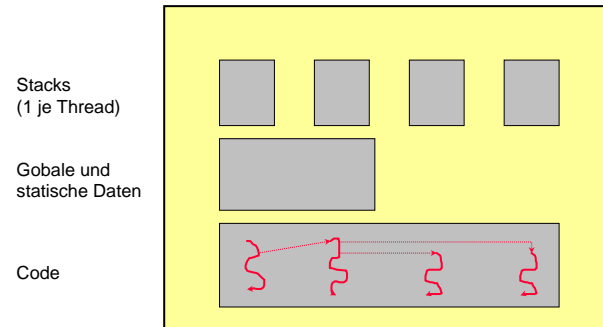
## Virtueller Adressraum eines NT-Prozesses



Der Scheduler von Windows – Konzepte und Strategien

Folie 5

## Ausführungspfade (Threads) in einem Prozess



Der Scheduler von Windows – Konzepte und Strategien

Folie 6

## Prozesse und Threads: Zusammenfassung

- ▶ Prozess: Umgebung und Adressraum für Threads
  - Ein Win32 Prozess enthält immer mindestens 1 Thread
- ▶ Thread: Code ausführende Einheit
  - Jeder Thread verfügt über einen eigenen Stack und Registersatz (insbesondere PC)
  - Threads bekommen vom Scheduler Rechenzeit zugeteilt
- ▶ Alle Threads sind Kernelmode Threads
  - Usermode-Threads möglich („Fibers“), aber unüblich
- ▶ Strategie von NT: Anzahl der Threads gering halten
  - keine blockierenden API-Aufrufe
  - Overlapped (asynchrones) IO
  - Thread-Pooling

Der Scheduler von Windows – Konzepte und Strategien

Folie 7

## Gliederung

1. Grundbegriffe
2. Eigenschaften des Schedulers
  - ▶ Grundlegende Eigenschaften
  - ▶ Prioritätenmodell
  - ▶ Dynamische Prioritätenanpassungen
3. Interner Aufbau
  - ▶ Interruptverarbeitung
  - ▶ Aufruf des Schedulers
4. Fazit

Der Scheduler von Windows – Konzepte und Strategien

Folie 8

## Grundlegende Eigenschaften des Schedulers

### 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 – 120 msec

## Prioritätsklassen und relative Threadpriorität

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

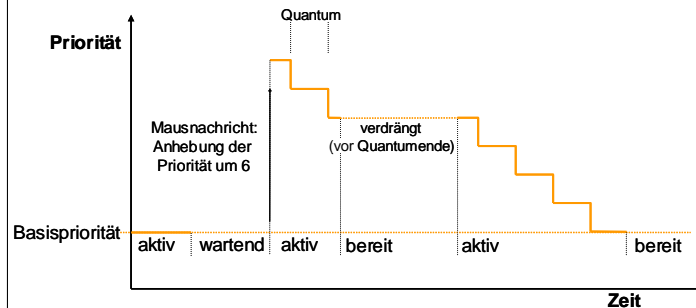
- Reines prioritätengesteuertes Round-Robin
  - Keine Fortschrittsgarantie
  - Keine dynamische Anhebung
  - Betriebssystem kann negativ beeinflusst werden
  - Spezielles Benutzerrecht erforderlich (SeIncraseBasePriorityPrivilege)
- Threadpriorität berechnet sich wie folgt:  
 $\text{REALTIME\_PRIORITY\_CLASS} + \text{Threadpriorität}$

## 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)

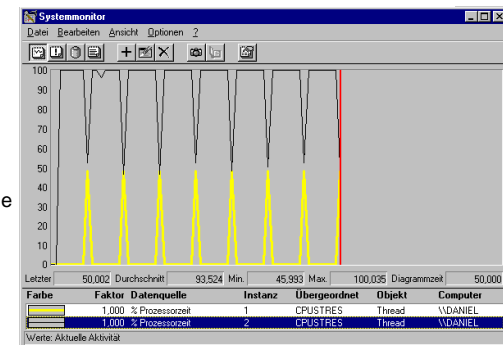
## Änderung der Priorität nach einem dynamic Boost



## Anhebung der Priorität durch Balance-Set-Manager

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

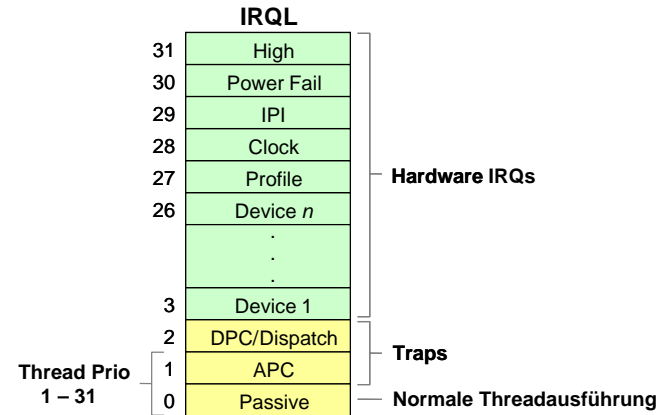
– Fortschrittsgarantie



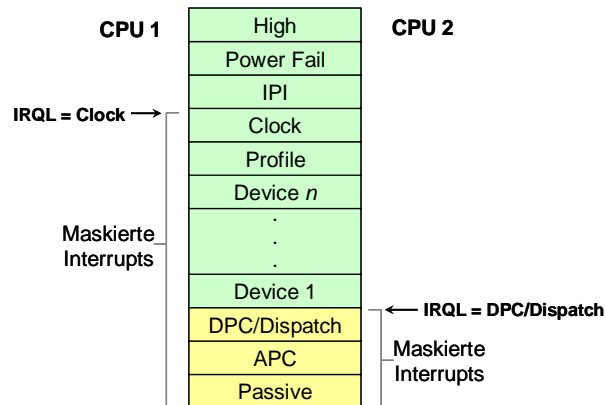
## Gliederung

1. Grundbegriffe
2. Eigenschaften des Schedulers
  - ▶ Grundlegende Eigenschaften
  - ▶ Prioritätenmodell
  - ▶ Dynamische Prioritätenanpassungen
3. Interner Aufbau
  - ▶ Interruptverarbeitung
  - ▶ Aufruf des Schedulers
4. Fazit

## Trap / Interruptverarbeitung



## Interruptverarbeitung auf SMP Systemen



## Aufruf des Schedulers

### Direkter Aufruf des Schedulers

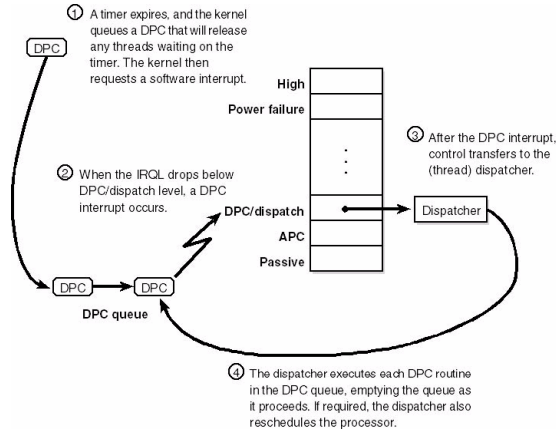
- ▶ Ende eines Threads (ExitThread())
- ▶ Freiwilligem Warten (Sleep(), WaitForXXX(), ...)

### Indirekter Aufruf des Schedulers

- ▶ Quantum abgelaufen
- ▶ Ein Thread höherer Priorität wird ablaufbereit

Indirekter Aufruf erfolgt durch DPCs und Traps

## Aufruf des Schedulers durch Traps



Der Scheduler von Windows – Konzepte und Strategien

Folie 21

## Auswahl des nächsten Threads (SMP)

Ziel: „gerechtes“ RoundRobin bei max. Durchsatz  
Problem: Cache-Effekte

Affinität (Zuordnung von CPUs zu Thread):

- ▶ **hard\_affinity:** Feste Zuordnung  
→ durch *SetThreadAffinity()*
- ▶ **ideal\_processor:** „Ideale“ Zuordnung  
→ bei Erzeugung zugewiesen („zufällig“)  
→ anpassbar 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

Der Scheduler von Windows – Konzepte und Strategien

Folie 22

## Auswahl des nächsten Threads (SMP)

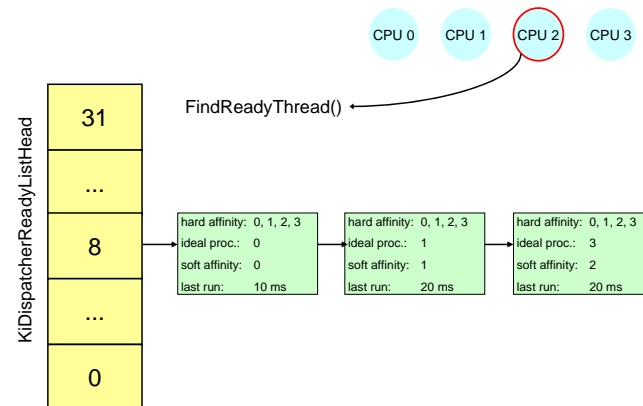
Algorithmus: CPU *n* ruft FindReadyThread() auf

- ▶ Wähle höchstpriorie, nicht-leere Warteschlange
- ▶ Suche in dieser Warteschlange nach Thread, mit
  - *soft\_affinity == n*
  - *ideal\_processor == n*
  - *currentTime() - last\_run > 2 Quantum*
  - *priority >= 24*
- ▶ sonst wähle Kopf der Warteschlange

Der Scheduler von Windows – Konzepte und Strategien

Folie 23

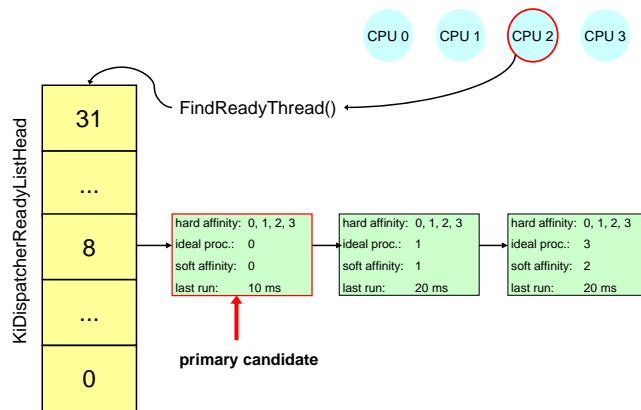
## Auswahl des nächsten Threads (SMP)



Der Scheduler von Windows – Konzepte und Strategien

Folie 24

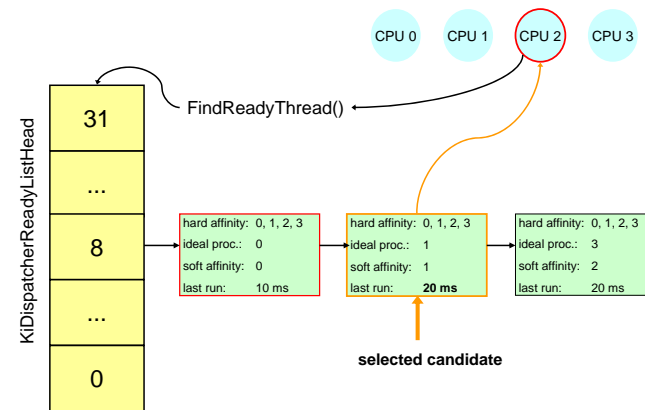
## Auswahl des nächsten Threads (SMP)



Der Scheduler von Windows – Konzepte und Strategien

Folie 25

## Auswahl des nächsten Threads (SMP)



Der Scheduler von Windows – Konzepte und Strategien

Folie 26

## 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

Der Scheduler von Windows – Konzepte und Strategien

Folie 27

## Fazit

Prioritätenmodell erlaubt keine Zuteilung der Prozessorzeit

- ▶ Dynamische Anpassungen beachten
- ▶ Usermode-Threads mit hohen Echtzeitprioritäten haben Vorrang vor allen System-Threads!
- ▶ Executive ist im allgemeinen unterbrechbar

Interruptverarbeitung

- ▶ Aufenthaltszeit des Systems in Interrupts bewusst klein gehalten.
  - Epiloge in DPCs ausgelagert
  - Längerfristige Arbeiten werden an Systemarbeitsthreads vergeben
- ▶ Weitere Verbesserungen für SMP in Windows 2003

Der Scheduler von Windows – Konzepte und Strategien

Folie 28