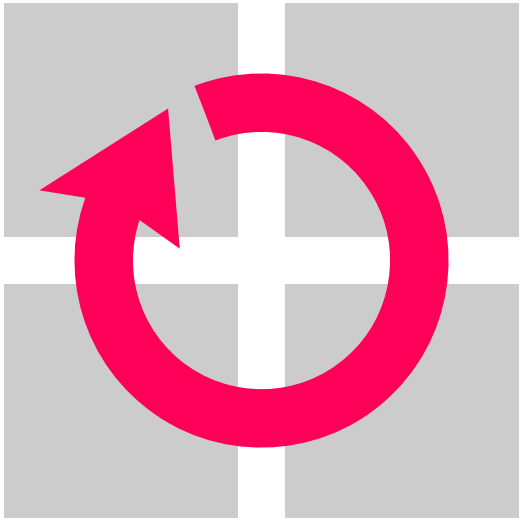
Resource Containers und ECOSystem

Andreas Weißel

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme
Universität Erlangen-Nürnberg

weissel@cs.fau.de



Überblick

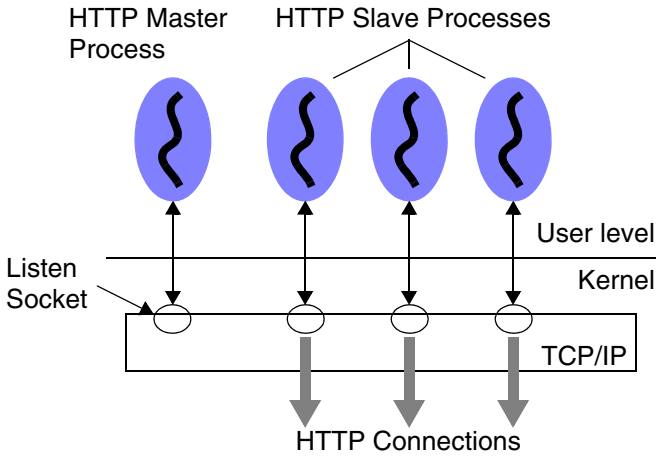
- Motivation - Ressourcen-Accounting
- Resource Containers
  - Anwendungen
  - Implementierung
  - Scheduling
- ECOSystem
  - Erfassung des Energieverbrauchs verschiedener Systemkomponenten
  - Anwendungen
- Zusammenfassung

Ressourcen-Accounting

- Accounting = Erfassen des Ressourcenverbrauchs
  - CPU-Zeit, Hauptspeicherbedarf, Festplatten-, Netzwerk-Bandbreite und -zugriffe, Energie usw.
  - Ziel: gleichmäßige, gerechte Ressourcenzuteilung ermöglichen, Ressourcenverbrauch begrenzen oder in Rechnung stellen, Garantie von Dienstgüte-Parametern
- Die bekannten Betriebssysteme unterstützen eine genaue Erfassung des Ressourcenverbrauchs nicht
  - Betriebssystem-Abstraktionen nicht ausreichend
- Beispiel Linux: nur eingeschränktes Accounting möglich
  - `time` liefert CPU-Zeit im User-space und Kern
  - verschiedene statistische Daten im `/proc`-Dateisystem (übertragene Dateisystemblöcke, ...)

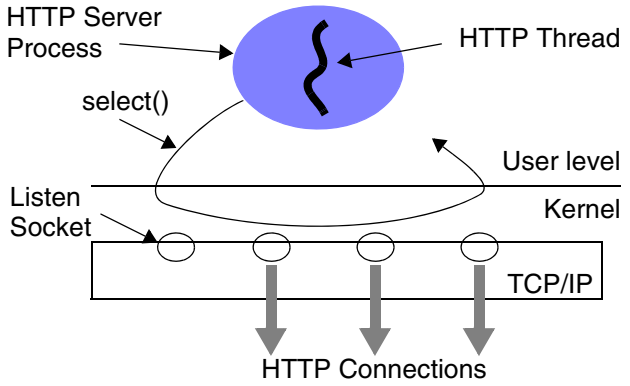
Beispiel HTTP-/Webserver

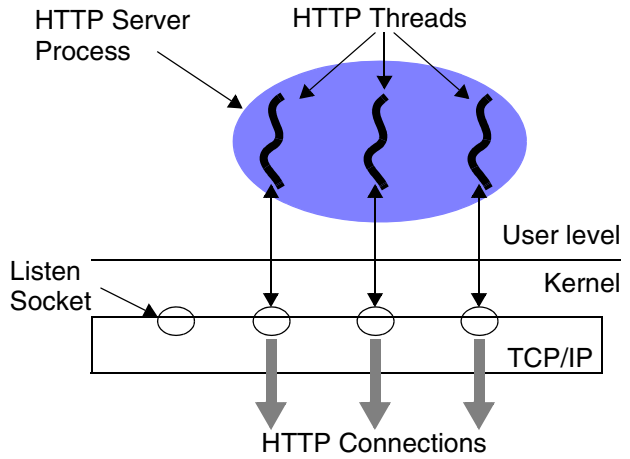
- Im Bereich der Server ist eine genaue und gerechte “Performance Isolation” von großer Bedeutung
  - vorhandene Betriebssystemmechanismen und -abstraktionen reichen dafür nicht aus
- Unterscheidung zwischen (unabhängigen) Aktivitäten und Prozessen/Threads
- Pre-forked Server
  - Zentraler Prozess nimmt Anfragen entgegen und gibt diese an Slave-Prozesse weiter
  - jede unabhängige Aktivität (HTTP-Verbindung) wird von einem *Prozess* ausgeführt



Beispiel HTTP-/Webserver

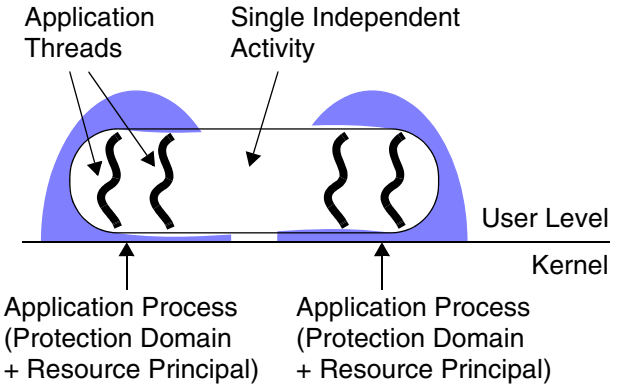
- Event-driven Server
  - Ein Prozess/Thread führt mehrere unabhängige Aktivitäten aus
  - Abfrage der Verbindungen über `select()`
- Single-Process multi-threaded
  - Ein Prozess mit mehreren Threads
  - jede unabhängige Aktivität (HTTP-Verbindung) wird von einem *Thread* ausgeführt
- weitere (Misch-)Formen möglich
  - z.B. Pre-forked multi-threaded





Beispiel HTTP-/Webserver

- Unter Umständen werden weitere Prozesse gestartet / aufgerufen
  - z.B. CGI-Programme
  - mehrere Prozesse arbeiten für eine unabhängige Aktivität
  - Erfassen des Ressourcenverbrauchs von Prozessen, die für einen Serverprozess arbeiten



- Prozesse bieten neben ihrer Funktion als Aktivitätsträger gleichzeitig eine isolierte Ablauf- & Schutzumgebung (*protection domain*)
  - Aktivitätsträger und Ablaufumgebung bedienen sich derselben Abstraktion

Beispiel HTTP-/Webserver

- Unabhängige Aktivitäten werden üblicherweise auf Prozesse abgebildet, obwohl diese nicht die geeignete Abstraktion darstellen.
  - keine Berücksichtigung asynchroner Ressourcennutzung
  - keine Berücksichtigung von Ressourcennutzung im Kern (CPU-Zeit und Netzwerkpuffer beim Empfang von Paketen)
- Beispiele

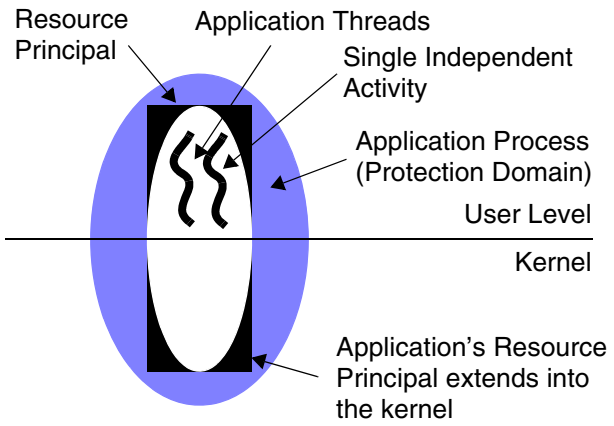
Anwendung	Zeit im user mode	Zeit im kernel
ftp (Datentransfer mit 92 Mbps)	10%	90%
Netscape (Download mit 320 KBps)	31%	69%
Squid proxy (120 Requests/s)	22%	78%
thttpd Webserver (400 Requests/s)	16%	84%
gcc (Kompiliervorgang über NFS)	79%	21%
Realplayer (Abspielen eines Video mit 30 Bilder/s)	32%	68%

Resource Containers

- Problem: “Performance Isolation” bei Serveranwendungen
  - Scheduler unterscheidet Prozesse; CPU-Zeit wird Prozessen zugerechnet; Prozesse als *Resource Principals*
  - Beispiel Verarbeiten eingehender Pakete:
    werden alle eingehenden Pakete gleichbehandelt, wird indirekt die Aktivität bevorzugt, die mehr Pakete als die anderen empfängt
  - Ressourcenverbrauch (z.B. Rechenzeit) kann oft nicht sauber/gerecht unter den Clients aufgeteilt werden
- *Resource Containers* als neue Betriebssystem-Abstraktion
  - Trennung zwischen isolierter Ablaufumgebung und Aktivitätsträger
  - abstrakte Betriebssystem-Einheit, die alle Ressourcen des Systems umfasst, auf die eine Anwendung zur Ausführung einer unabhängigen Aktivität zurückgreift
  - Accounting von Kern-Ressourcen und asynchronen Operationen

Anwendungen

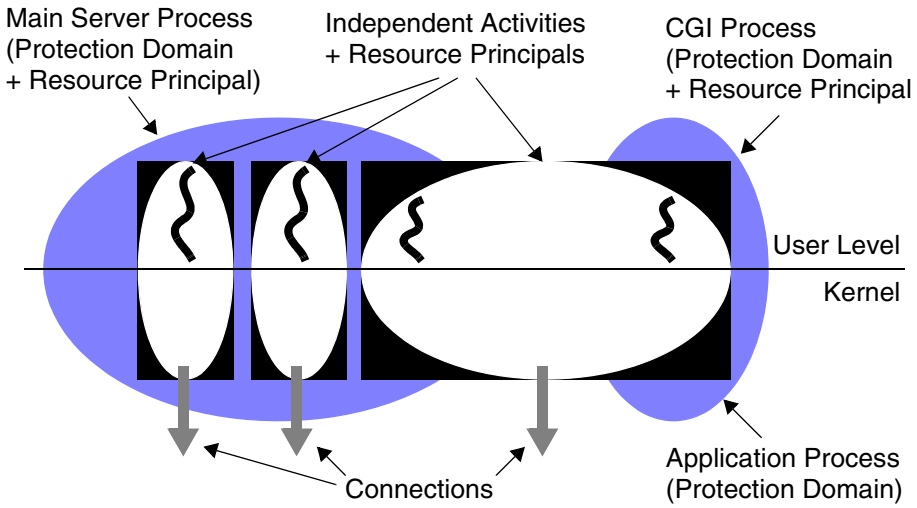
- Erfassen der verbrauchten CPU-Zeit (sowohl im User-Space als auch im Kern)
- Erfassen des Verbrauchs von Objekten des Kerns (Sockets, Netzwerkpuffer)



- Unterscheidung verschiedener Aktivitäten, z.B. Netzwerkverbindungen mit verschiedenen Clients
- Einbeziehung dieser Informationen in Scheduling-Entscheidungen

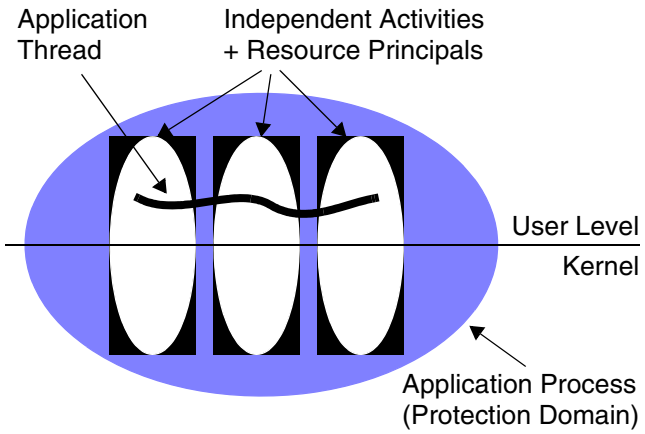
Anwendungen

- Mehrere Threads in einem Prozess als unabhängige Aktivitätsträger
- Mehrere Prozesse oder Threads in mehreren Prozessen als ein Aktivitätsträger (z.B. CGI-Prozesse)



Resource Containers

- Ein Thread führt mehrere unabhängige Aktivitäten aus
- z.B. Event-driven Server unter Verwendung von `select()`



Implementierung

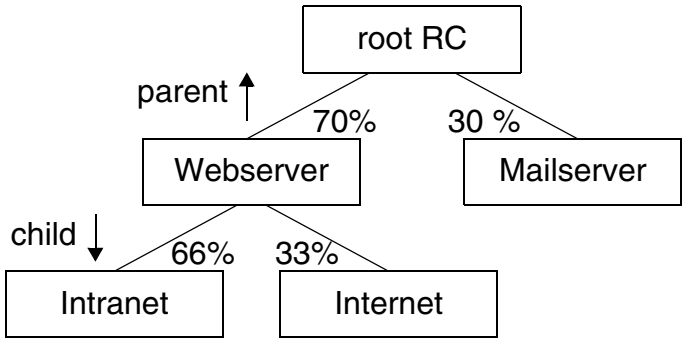
- Thread-Verwaltungsstruktur enthält Referenz auf Resource Container-Struktur (RC)
- Attribute von RCs: Scheduling-Parameter, Ressourcen-Limits, QoS-Parameter, Zugriffskontrolle
- Accounting-Informationen können für Scheduling-Entscheidung herangezogen werden
- Dynamische Bindung zwischen Threads und Resource Container (*resource binding*)
  - geerbt vom erzeugenden Prozess
  - Wechsel der Resource Container möglich
  - z.B. Multiplexen zwischen verschiedenen Netzwerk-Verbindungen

Scheduling

- Assoziieren von Scheduling-Information (z.B. CPU-Zeit) mit einer Aktivität, nicht mit einem Thread/Prozess
- Garantie von Dienstgüte-Parametern möglich (z.B. CPU-Zeit von mindestens 20%).
- Verschiedene Scheduling-Modelle möglich
  - numerische Priorität
  - garantierter Anteil an der CPU-Zeit
  - Limitieren der verfügbaren CPU-Zeit einer Aktivität
- *Scheduling Binding*: Bindung zwischen einem Thread und einer Menge von Containern
  - Scheduling-Entscheidung anhand des zusammengefassten Ressourcenverbrauchs und der Priorität der Container

Resource Container Hierarchie

- Ressourcen-Verbrauch wird durch die Scheduling-Parameter der Eltern-Container beschränkt
- Einer Aktivität kann eine Ressourcen-Menge garantiert (z.B. mind. 70% CPU-Zeit), oder der Ressourcenverbrauch limitiert werden (max. 70%).



Resource Container Hierarchie

- Eine Aktivität kann ihre Ressourcen frei verwalten
  - Erzeugen mehrerer untergeordneter Resource Container
  - verfügbare Ressourcen beliebig aufteilbar
  - eigenständige Steuerung und Kontrolle des Ressourcenverbrauchs durch die Aktivitäten selbst möglich
  - Resource Container können allerdings ihre eigene Ressourcenzuweisung bzw. -begrenzung nicht beeinflussen.
- Analogie zur Userspace-Threadverwaltung

Operationen auf Resource Containers

- Erzeugen: entweder explizit oder bei `fork()`
  - Resource Container werden von den Prozessen über Dateideskriptoren angesprochen
  - diese werden bei `fork()` automatisch vererbt
- Bindung lösen: `close()` auf den Dateideskriptor
- Resource Container lassen sich mehreren Prozessen zur Verfügung stellen
  - Verschicken der Deskriptoren über `sendmsg()` & `recvmsg()`
- Binden eines Datei- oder Socket-Deskriptors an einen Resource Container
  - Ressourcenverbrauch (Dateizugriffe, Netzwerk-Puffer) wird über diesen Container abgerechnet.

Überblick

- Motivation - Resource Accounting
- Resource Containers
  - Anwendungen
  - Implementierung
- ECOSystem
  - Erfassung des Energieverbrauchs verschiedener Systemkomponenten
  - Anwendungen
- Zusammenfassung

ECOSystem

- Einheitliche Abrechnung des Energieverbrauchs verschiedener Hardware-Komponenten
  - basierend auf Resource Container
  - ➜ benutzerdefinierte Aufteilung der verfügbaren Energie unter den Anwendungen
  - ➜ Garantie einer bestimmten Batterielaufzeit durch Steuerung der Entlade-Rate
- Einheit für Allokation von Ressourcen und Abrechnung von Ressourcenverbrauch: *Currentcy* (10 mJ)
- erlaubt Anwendungen, eine bestimmte Energiemenge innerhalb einer bestimmten Zeit zu verbrauchen

ECOSystem / Currentcy

- Systemweite Zuweisung von Currentcy
  - Höhe der Zuweisung pro Epoche konfigurierbar
  - Epochenlänge konfigurierbar
    - zu lang: periodische Spitzen im Stromverbrauch ("bursts")
    - zu kurz: u.U. nicht genug Currentcy, um komplexe Aufgaben durchzuführen
- Prozessspezifische Currentcy-Zuweisung
  - entspricht der Priorität und/oder der verfügbaren Energiemenge des Prozesses
  - Wie soll mit nicht verwendetem Currentcy umgegangen werden?
    - verwerfen
    - in die nächste Epoche retten (evtl. begrenzt auf Maximalwert)
    - auf andere Prozesse verteilen
  - Überziehen möglich (negativer Currentcy-Wert)

Accounting verschiedener Komponenten

- CPU (Pentium III)
  - Timer-Interrupt alle 10 ms: Verbrauch von 15.55 Einheiten (155.5 mJ)
  - Prozesse verbrauchen ihr Currentcy ohne Rücksicht auf die Länge der Epoche
  - periodische Spitzen im Stromverbrauch zu Beginn jeder Epoche
  - ➜ Glätten des Currentcy-Verbrauchs notwendig
- Netzwerk-Interface
  - E_{send} = (versendete Bits x Stromverbrauch beim Senden)/Datenrate
  - $E_{receive}$ = (empfangene Bits x Stromverbrauch beim Empfangen)/ Datenrate

Accounting verschiedener Komponenten

- Festplatte
  - Kosten einer Lese-/Schreiboperation:

$$\frac{\text{Stromverbrauch im aktiven Zustand (W)}}{\text{Übertragungsrate (KB/s)}} \cdot \text{Puffergröße (KB)}$$

  - Energieverbrauch beim Zugriff auf einen Block: 1.65 mJ
  - bei mehreren Schreibzugriffen auf einen Block im Buffer Cache wird nur der letzte Zugriff berechnet
  - Moduswechsel: 3000 mJ (2 Sekunden)
    - Kosten werden auf alle Prozesse verteilt, die zwischen dem Wechsel Standby→Idle und Idle→Standby auf die Festplatte zugreifen
    - Anteil entsprechend Zahl der gelesenen/geschriebenen Blöcke

Strategien zur Verwendung von Currentcy

- Ziel: Currentcy Conserving Scheduling
  - Abweichung zwischen zugeteiltem und tatsächlichem Currentcy-Verbrauch feststellen
  - Gesamte Currentcy-Menge erhöhen, falls tatsächlicher Stromverbrauch geringer als erwartet
  - nicht verwendetes Currentcy
    - ein Teil wird dem Prozess in der nächsten Epoche zugerechnet
    - Rest auf andere Prozesse verteilen
  - ➜ Erhöhen der Currentcy-Zuweisung bei den Prozessen, die Currentcy effektiv einsetzen (die energie-effizient arbeiten)

Strategien zur Verwendung von Currentcy

- Ziel: Proportional Energy Use
  - Erweiterung des Linux-Scheduler
    - Überprüfung, ob Prozess noch genug Currentcy zur Verfügung hat
  - Energiebewusster Scheduler
    - Scheduling-Entscheidung aufgrund des Energieverbrauchs eines Tasks
    - Scheduling-Priorität: Currentcy-Zuweisung dividiert durch tatsächlicher Verbrauch innerhalb der Periode
  - ➜ interaktive Prozesse werden bevorzugt
  - ➜ zusätzliche Glättung des Currentcy-Verbrauchs möglich

Strategien zur Verwendung von Currentcy

- Ziel: Koordinierung mehrerer Geräte
  - Einhaltung des Energielimits schwierig beim Empfang von Paketen
  - Bandbreite der Netzwerk-Verbindung reduzieren, falls zugehöriger Prozess seine Currentcy-Menge verbraucht hat
  - alle Verbindungen erhalten Bandbreite proportional zum Energie-Anteil der zugehörigen Prozesse
  - Implementierung: Steuerung der Bandbreite über die TCP-Window-Größe (Verkleinern falls Bandbreite reduziert werden soll)
  - ➜ in Kombination mit energiebewusstem Scheduler sowohl gerechte Zuteilung der Bandbreite als auch des Energieverbrauchs möglich

Strategien zur Verwendung von Currentcy

- Ziel: Energieverbrauch glätten
  - Epoche verkleinern
  - Self-pacing im EC-Scheduler
  - Einen Task verzögern, falls Currentcy zu “schnell” innerhalb einer Epoche verbraucht wird
  - Fortschritt berechnen:
    Currentcy-Menge, die bis jetzt verbraucht wurde geteilt durch die Zuweisung für diese Epoche

Strategien zur Verwendung von Currentcy

- Ziel: Energie-Effizienz
  - z.B. Energiebewusste Zugriffssteuerung für Festplatten:
    - Prozesse belohnen, die Zugriffe “en block” durchführen:
    - hoher Einstiegspreis: Kosten für Zugriff im Standby-Modus viel höher als die tatsächlichen Kosten
    - nur die tatsächlichen Kosten werden berechnet
    - Prozess hat genügend Currentcy, um lange zu laufen und während dieser Zeit Festplattenzugriffe durchzuführen
    - lange Idle-Zeiten vermeiden
  - Kooperation mehrerer Prozesse, um Currentcy zu sparen (“Cost-sharing”)

Zusammenfassung

- Resource Containers
  - Trennung zwischen isolierter Ablaufumgebung und Aktivitätsträger
  - abstrakte Betriebssystem-Einheit, die alle Ressourcen des Systems umfasst, auf die eine Anwendung zur Ausführung einer unabhängigen Aktivität zurückgreift
  - Accounting von Kern-Ressourcen und asynchronen Operationen
  - Implementierung auf CPU-Rechenzeit beschränkt
- ECOSystem
  - Resource Container-Implementierung zur Erfassung und Steuerung des Energieverbrauchs