LINUX KERNEL

INTERNA

QUARITSCH MARKUS
WINKLER THOMAS

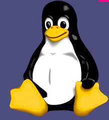
WS 2003/04

slides created with LaTeX 2$_\varepsilon$

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

INHALT

- Scheduling
- Memory Management (auszugsweise)
- Systemaufrufe
- Netzwerk-Stack (auszugsweise)
- Sicherheit: Linux Security Modules



Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

UMFANG DER KERNEL-SOURCEN

- Archivgröße (tar.gz):

  06/1996: 2.0: 5 MiB

  01/1999: 2.2: 10 MiB

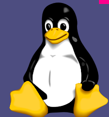
  01/2001: 2.4: 20 MiB

  12/2003: 2.6: 32 MiB

- Statistische Daten (2.6):

6229	Header Dateien (*.h*)
6194	C Quelldateien (*.c*)
674	Assembler-Dateien (*.S*)
663	Makefiles (*Makefile*)

- $\sim$ 4 Millionen Zeilen Code (ohne Kommentar)



Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

- Aufteilung entsprechend Aufgaben
  - `kernel`: zentrale Bestandteile, ca. 20 kLoC
  - `mm`: High-Level Speicherverwaltung, ca. 15 kLoC
  - `fs`: Implementation unterstützter Dateisysteme
  - `net`: Implementation unterstützter Netzwerkprotokolle
  - `drivers`: Treiber für unterstützte Hardware (ca. 87 MiB)
  - `include`: exportierte Header-Dateien
  - `crypto`: cryptographische Funktionen
  - `arch`: Architekturabhängige Funktionen

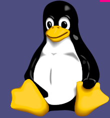
© 2004 Quaritsch Markus, Winkler Thomas

SCHEDULING

AUFGABEN

- Rechenzeit fair zwischen den Prozessen aufteilen
- Prozesse sind mit Priorität versehen
- seit Kernel 2.6 O(1) Scheduler

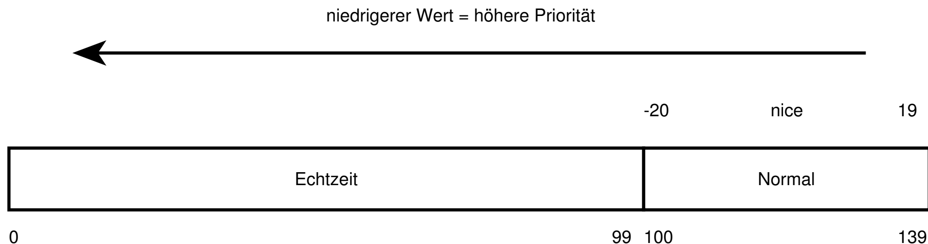
© 2004 Quaritsch Markus, Winkler Thomas

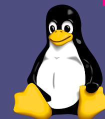


Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

PROZESSPRIORITÄTEN

- statische Priorität (nice): -20 bis +19 (= niedrigste Priorität)
- zusätzliche Prioritäten für Echtzeitprozesse
- Prioritäten im Kernel von 0 bis 139
  - 0 bis 99: Echtzeitprozesse
  - 100 bis 139: nice Werte von -20 bis +19





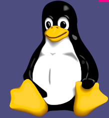
LINUX KERNEL
INTERNA

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

TIMESLICES

- Timeslice ist Zahlenwert der angibt wie lange ein Task laufen darf bis ihm der Prozessor entzogen wird (der Task „preempted“ wird)
- im Bereich von 10ms bis 200ms
- wird in Abhängigkeit von der Priorität berechnet
- Task muss seine Timelice nicht auf einmal verbrauchen sondern kann vorzeitig die Kontrolle abgeben (z.B. warten auf I/O)

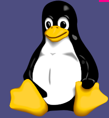


Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Datenstrukturen

- Prozess Descriptor: `struct task_struct`
  - Informationen über Task, unter anderem auch für Scheduling
  - `prio` dynamische Priorität
  - `static_prio` statische Priorität
  - `sleep_avg` sagt aus wie oft und lange ein Prozess geschlafen hat
  - `policy` Scheduling Policy (Priority, FIFO, RR)
  - `time_slice` verbleibende CPU-Zeit des Tasks



© 2004 Quaritsch Markus, Winkler Thomas

```
1  struct task_struct {
2      /* ... */
3
4      int prio, static_prio;
5      struct list_head run_list;
6      prio_array_t *array;
7
8      unsigned long sleep_avg;
9      long interactive_credit;
10     unsigned long long timestamp;
11     int activated;
12
13     unsigned long policy;
14     cpumask_t cpus_allowed;
15     unsigned int time_slice, first_time_slice;
16
17     /* ... */
18 };
```

Listing 1: Scheduling Informationen in `task_struct` (*include/linux/sched.h*)

© 2004 Quaritsch Markus, Winkler Thomas

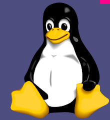
Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

eine Runqueue pro Prozessor

- `active` Array: alle Prozesse mit Timeslice größer 0
- `expired` Array: alle Prozesse die ihre Timeslice aufgebraucht haben
- `nr_running` Anzahl der lauffähigen Prozesse
- `curr` aktuell laufender Prozess

```
1  struct runqueue {
2      spinlock_t lock;
3      unsigned long nr_running, nr_switches,
4                         expired_timestamp, nr_uninterruptible;
5      task_t *curr, *idle;
6      struct mm_struct *prev_mm;
7      prio_array_t *active, *expired, arrays[2];
8      int prev_cpu_load[NR_CPUS];
9      /* ... */
10     task_t *migration_thread;
11     struct list_head migration_queue;
12     atomic_t nr_iowait;
13 };
```

Listing 2: `runqueue` Datenstruktur (*kernel/sched.c*)

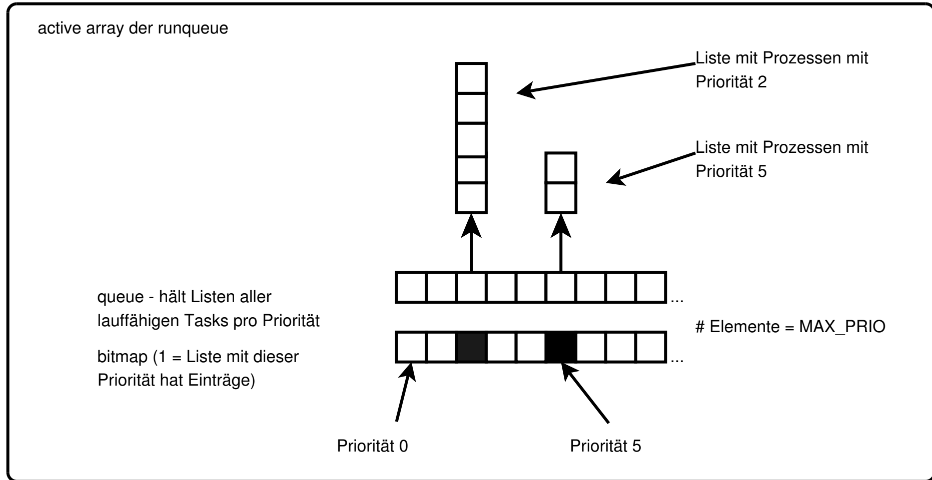


Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

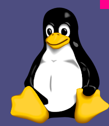
© 2004 Quaritsch Markus, Winkler Thomas

```
1 struct prio_array {
2       int nr_active;
3       unsigned long bitmap[BITMAP_SIZE];
4       struct list_head queue[MAX_PRIO];
5 };
```

Listing 3: `prio_array` Datenstruktur (*kernel/sched.c*)



© 2004 Quaritsch Markus, Winkler Thomas



- Inhalt
- Statistische Daten
- Scheduling
- Memory Management
- Systemaufrufe
- Netzwerk
- Sicherheit

Scheduler

- auswählen welcher Task als nächstes laufen soll
  - erstes gesetzte Bit in bitmap finden
  - ersten Task aus Liste des zugehörigen `queue` Elements auswählen
- Kernel 2.4:
  - Prozesse nicht nach Priorität sortiert gespeichert
  - keine Trennung zwischen `active` und `expired`
  - für Auswahl des nächsten Prozesses über ganze Liste iterieren (O(n))

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

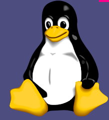
© 2004 Quaritsch Markus, Winkler Thomas

Aufbrauchen der Timeslice (2.4)

- Timeslices immer dann neu berechnet wenn sie für alle Tasks verbraucht ist
- Neuberechnung erfolgte als Schleife über alle Tasks – O(n) Zeit

Aufbrauchen der Timeslice (2.6)

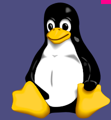
- wenn Timeslice eines Tasks den Wert 0 erreicht, dann wird er vom `active` ins `expired` Array verschoben und gleichzeitig die Timeslice neu berechnet
- wenn alle Timeslices 0: einfacher Pointer Tausch (`active` $\leftrightarrow$ `expired`)



Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

- 2 Arten von Tasks:
  - stark I/O lastig (z.B. Texteditor)
  - stark CPU lastig (z.B. Compiler)
- erfordern unterschiedliche Behandlung durch Scheduler
- Anpassung der dynamischen Priorität und der Timeslice

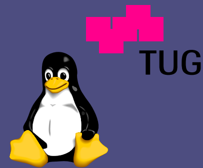


Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Neuberechnung von Priorität und Timeslice

- Funktion `effective_prio` berechnet Bonus (-5 bis +5) basierend auf `sleep_avg` $\longrightarrow$ dynam. Priorität
- `sleep_avg` ist Maß für Interaktivität des Tasks
  - wenn Task aufwacht wird die Dauer die er geschlafen hat zu `sleep_avg` dazu addiert
  - bei jedem Timer-Tick wird `sleep_avg` dekrementiert
  - I/O lastige Tasks haben höhere `sleep_avg` als CPU lastige Tasks
- Berechnung der Timeslice basiert auf dynamischer Priorität des Prozesses
- hoch interaktive Tasks können statt in `expired` wieder in `active` eingereiht werden

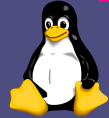


Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Kernel Preemption

- bisher nicht möglich einem Task die CPU zu entziehen wenn er Kernel-Code ausgeführt hat
- seit 2.6 auch Preemption für Kernel-Code möglich wenn keine Locks gehalten werden
- eigener Counter für gehaltene Locks
- ist bei Rückkehr aus Interrupt Handler der `preempt_count` auf 0 oder wird dieser auf 0 dekrementiert (alle Locks freigegeben) wird der Scheduler aufgerufen
- vor 2.6 nur kooperative Preemption im Kernel durch expliziten Aufruf des Schedulers (Entwickler für sicheren Zustand verantwortlich)



Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

PERFORMANCE

- Vergleich der OSDL Labs zwischen 2.4 und 2.6

Hackbench:
Performance for Process Groups - 1 CPU

Avg. Time (sec) - small is good

Number of Processes

'2.4.18-3'
'2.6.0-test9'

0, 20, 40, 60, 80, 100, 120
20, 40, 60, 80, 100, 120, 140, 160, 180, 200

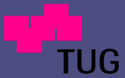
Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Memory Management

Aufgaben

- Verwaltung des physikalischen Speichers
- Verwaltung des virtuellen Speichers für jeden Prozess

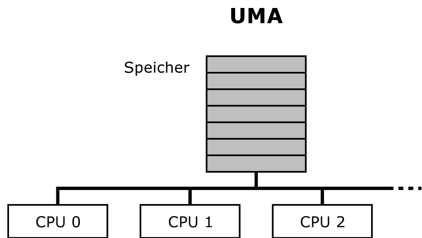


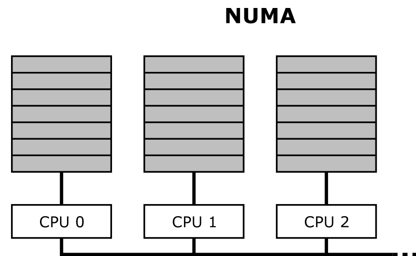
Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Speichermodell

- Klassifizierung in
  - Uniform Memory Access (UMA)
  - Non Uniform Memory Access (NUMA)



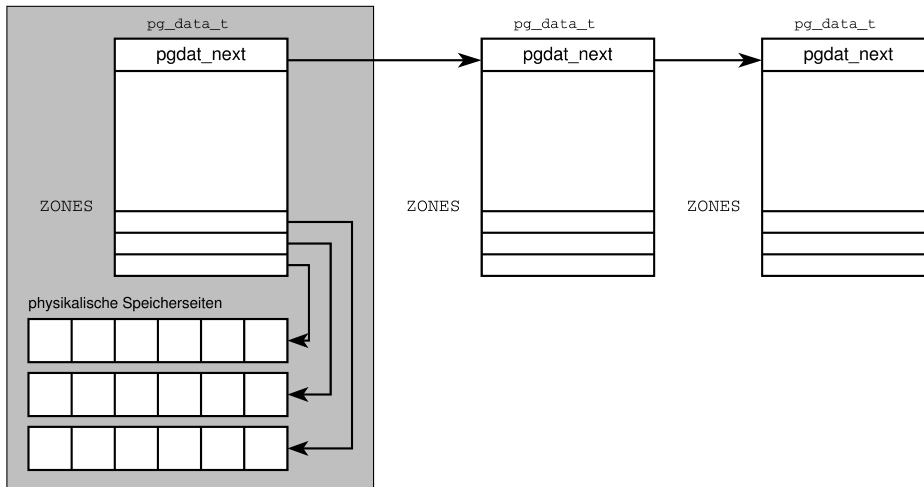


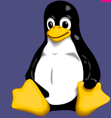
Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Abbildung in Datenstrukturen

- `pg_data_t` repräsentiert eine Node
- Aufteilung einer Node in bis zu 3 Zonen (DMA, Highmem, Normal)





LINUX KERNEL INTERNA

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Repräsentation einer Node

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit


```
1  typedef struct pglist_data {
2          struct zone node_zones[MAX_NR_ZONES];
3          struct zonelist node_zonelists[MAX_NR_ZONES];
4          int nr_zones;
5          struct page *node_mem_map;
6          unsigned long *valid_addr_bitmap;
7          struct bootmem_data *bdata;
8          unsigned long node_start_pfn;
9          unsigned long node_present_pages;
10         unsigned long node_spanned_pages;
11         int node_id;
12         struct pglist_data *pgdat_next;
13         wait_queue_head_t       kswapd_wait;
14 } pg_data_t;
```

Listing 4: Repräsentation einer Node mittels `pg_data_t` (*include/linux/mmzone.h*)

© 2004 Quaritsch Markus, Winkler Thomas

FALLBACK-LISTE

- Nicht genügend Speicher in gewünschter Zone
- Ausweichen auf andere Zonen $\longrightarrow$ Fallback-Liste
  - Enthält Zeiger auf alle weitere Zonen gereiht nach Kosten

DMA	C0	D0	A0	B0	NULL								
Normal	C1	C0	D1	D0	A1	A0	B1	B0	NULL				
HighMem	C2	C1	C0	D2	D1	D0	A2	A1	A0	B2	B1	B0	NULL

0 ... DMA
1 ... Normal
2 ... HighMem

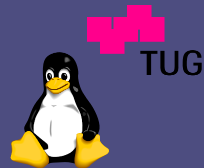
A ... Node 0
B ... Node 1
C ... Node 2
D ... Node 3

LINUX KERNEL INTERNA
Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Buddy-System

- Algorithmus für Verwaltung freier Speicherseiten
- Nur für Seiten einer Zone
- Zusammenhängende Seiten in 12 Gruppen eingeteilt
  - Gruppen mit: 1, 2, 4, . . . 2048 Seiten (4 kiB – 8 MiB)
  - Startadresse muss Vielfaches der Gruppengröße sein: z.B. für Block mit 16 Seiten: Vielfaches 16×2^{12}
- Freie Speicherblöcke (Buddies) möglichst zusammenfassen
- Nur ganzzahlige Zweierpotenzen von Speicherseiten allozierbar



Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

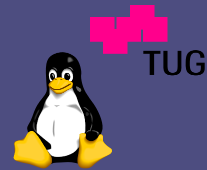
Datenstruktur

```
1  struct zone {
2      /* ... */
3      struct free_area    free_area[MAX_ORDER];
4      /* ... */
5  };
6
7  struct free_area {
8      struct list_head    free_list;
9      unsigned long       *map;
10 };
```

Listing 5: Datenstrukturen für das Buddy-System (*include/linux/mm.h*)

- `free_list`: Liste mit freien Speicherblöcken
- `map`: Bitmap für Zustand der Buddy-Paare (je Paar ein Bit)
  - **Bit gesetzt:** Beide Buddies vergeben oder frei
  - **Bit ungesetzt:** Einer der Buddies ist frei

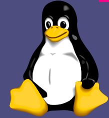
© 2004 Quaritsch Markus, Winkler Thomas



Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

INTERFACE DES BUDDY-SYSTEMS

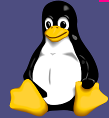
- `buffered_rmqueue`:
  - Entnimmt Speicherblock aus `free_list`
  - evtl. aufspalten größerer Blöcke
- `__free_pages`:
  - Einfügen von freigegebenem Speicher in die Datenstruktur
  - Zusammenfügen von Buddies



© 2004 Quaritsch Markus, Winkler Thomas

Allokation von Speicher

- Hauptfunktion: `__alloc_pages`
  - Durchsucht Fallback-Liste ob genügend freier Speicher
  - Versucht mittels `buffered_rmqueue` zusammenhängenden Speicher zu finden
- Inkrementelles Minimum
  - Verhindert starke Belastung von 'benachbarten' Zonen

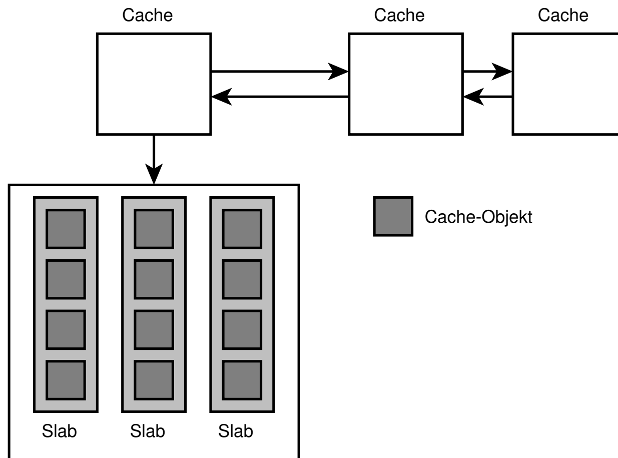


Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

SLAB-ALLOKATOR

- Effiziente Verwaltung von kleineren Speicherbereichen (Größenordnung einige Bytes)
- Häufig benutzte Datenstrukturen werden in einem Cache gehalten
  - Für jede Datenstruktur ein eigener Slab-Allokator
  - Bessere Ausnutzung von Hardware-Caches



© 2004 Quaritsch Markus, Winkler Thomas

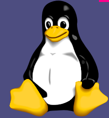
- Cache
  - Verwaltung der zugehörigen Slabs
  - 3 Listen mit Slabs (frei, teilweise belegt, belegt)
  - Cache der zuletzt freigegebenen Objekte für jede CPU
- Slab
  - Information zur Verwaltung der Objekte auf dem Slab
  - Menge von Objekten
- per CPU Cache

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

INTERFACE

- Speicher anfordern
  - `kmem_cache_alloc`: Speicher für ein bestimmtes Objekt anfordern
  - `kmalloc`: Analogon zu `malloc` der C-Bibliothek
- Speicher freigeben
  - `kmem_cache_free`
  - `kfree`



Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

SYSTEMAUFRUFE

- Benutzerprogramme werden im Usermode ausgeführt
  - Kein direkter Zugriff auf Ressourcen
  - Zugriff auf Ressourcen über Service-Routinen des Kernel

$\Rightarrow$ System-Calls

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

STANDARDS

- Ermöglichen leichtere Portierbarkeit
- Mehrere Standards definiert
  - POSIX (Portable Operating System Interface)
  - System V
  - 4.3 BSD
- Linux bemüht sich, POSIX-Standard zu implementieren
- Andere Standards teilweise implementiert

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Vorhandene Systemaufrufe

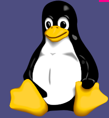
- Über 200 Systemaufrufe (architekturabhängig)
  - Prozessverwaltung
  - Zeitoperationen
  - Signalverarbeitung
  - Scheduling
  - Dateisystem
  - Speicherverwaltung (C-Standardbibliothek)
  - Interprozesskommunikation & Netzwerkfunktionen
  - Systeminformationen und -einstellungen
- Systemaufrufe werden über eindeutige Kennzahl angesprochen (architekturabhängig)

- Inhalt
- Statistische Daten
- Scheduling
- Memory Management
- Systemaufrufe
- Netzwerk
- Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Realisierung von Systemaufrufen

1. Anwendung ruft Funktion aus C-Standardbibliothek auf
2. C-Bibliothek speichert Parameter in Register
3. Auslösen eines Software-Interrupts
4. Interrupt-Serviceroutine (Teil des Kernels) wechselt in Kernelmode
5. Delegiert Systemaufruf an zuständige Funktion (Systemcall-Handler)
6. Return-Wert wird über register an C-Bibliothek zurückgegeben

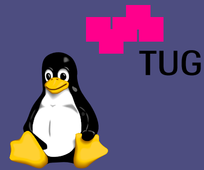


Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

EINSCHRÄNKUNGEN

- Nummer des Systemaufrufs wird über Register `eax` übergeben
- Parameterübergabe mittels Register
  - Nur primitive Datentypen
  - max. 5 Parameter möglich
  - komplexe Datentypen per Referenz
- Kein direkter Zugriff auf Prozessspeicher möglich

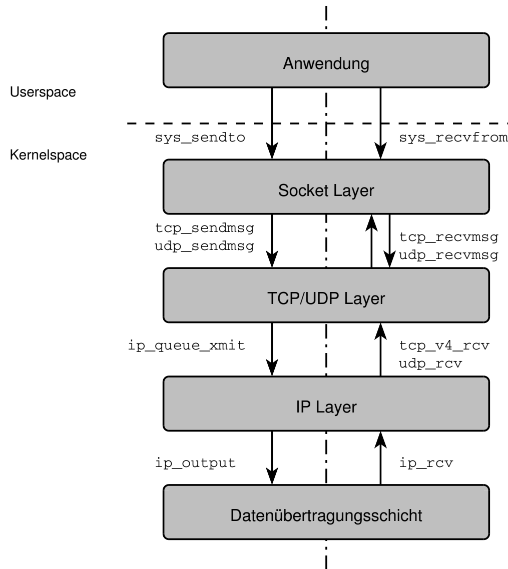


Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

NETZWERK

- gängige Schichtmodelle spiegeln sich auch im Kernel wieder



LINUX KERNEL
INTERNA

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Socket Layer

- Vermittlungsschicht zwischen Anwendung (Userspace) und TCP-Stack
- Unterschiedliche Arten von Sockets
  - SOCK_STREAM
  - SOCK_DGRAM
  - SOCK_RAW
- Zeiger auf Funktionspointer zur Transportschicht

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

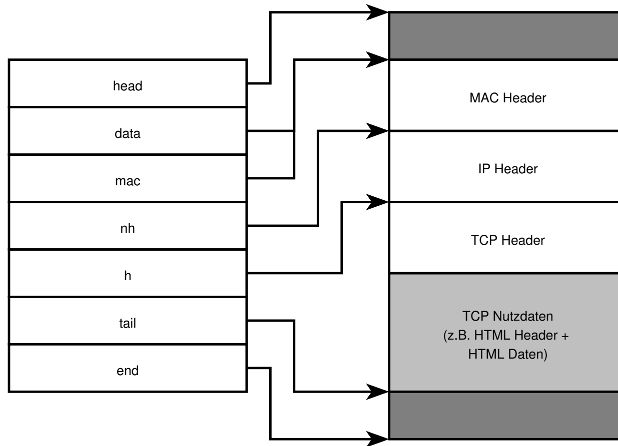
DATENSTRUKTUREN

- Netzwerkpakete möglichst effizient zwischen den Schichten weiterreichen
- pro Schicht in der Regel einen eigenen Header
- Datenstruktur `struct sk_buff`
- Übergang zwischen Schichten einfache Änderung der Pointer
  - `next` und `prev` dienen zur Verwaltung der `sk_buff` Instanzen in einer doppelt verketteten Liste
  - `head` und `end` zeigen auf Anfang bzw. Ende des Speicherbereichs, der für das Paket reserviert ist

LINUX KERNEL
INTERNA

- Inhalt
- Statistische Daten
- Scheduling
- Memory Management
- Systemaufrufe
- Netzwerk
- Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas



- `data` und `tail` zeigen auf Anfang bzw. Ende des Protokolldatenbereiches
- `mac` zeigt auf den Beginn des MAC Headers
- `nh` zeigt auf den Beginn des Headers der Vermittlungsschicht (z.B. IP)
- `h` zeigt auf den Beginn des Headers der Transportschicht (z.B. TCP)

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

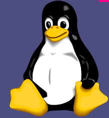
Transportschicht

- Üblicherweise 2 Protokolle:
  - TCP (Transmission Control Protocol)
  - UDP (User Datagram Protocol)

Eingehende Pakete

- `tcp_v4_rcv`
- Aufteilung in 2 Gruppen:
  - leicht verarbeitbar
    - Empfangsbestätitung
    - als nächstes erwartete Daten
    - kein Flag (SYN, URG, RST, FIN) gesetzt
  - aufwändiger zu verarbeiten
- Anwendung über eingetroffene Daten benachrichtigen

© 2004 Quaritsch Markus, Winkler Thomas



- Inhalt
- Statistische Daten
- Scheduling
- Memory Management
- Systemaufrufe
- Netzwerk
- Sicherheit

Versenden von Daten

- `tcp_send_msg` (Zustand TCP_ESTABLISHED)
- Daten aus Userspace kopieren, verarbeiten und an Vermittlungsschicht übergeben (`ip_queue_xmit`)

- Inhalt
- Statistische Daten
- Scheduling
- Memory Management
- Systemaufrufe
- Netzwerk
- Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

VERMITTLUNGSSCHICHT

Transportschicht (z.B. TCP)

ip_local_deliver

ip_queue_xmit

Netfilter:
IP_LOCAL_INPUT

Netfilter:
IP_LOCAL_OUTPUT

Routing

Weiterleitung
ip_forward

Routing

Netfilter:
IP_PRE_ROUTING

Netfilter:
IP_POST_ROUTING

ip_rcv

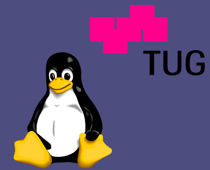
Netfilter:
IP_FORWARD

ip_output

dev_queue_xmit

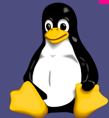
Datenübertragungsschicht

© 2004 Quaritsch Markus, Winkler Thomas



Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

DATENÜBERTRAGUNGSSCHICHT



- Übertragung der Pakete
- Enge Zusammenarbeit mit Netzwerkkarte (repräsentiert als `net_device`)
- Empfang von Paketen über Interrupt-Behandlungsroutine
- Daten in `sk_buff`-Instanz legen
- Senden von Paketen über Funktion des Netzwerkkarten Treibers

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Userspace

Kernelspace

Anwendung

sys_sendto

sys_recvfrom

Socket Layer

tcp_sendmsg
udp_sendmsg

tcp_recvmsg
udp_recvmsg

TCP/UDP Layer

ip_queue_xmit

tcp_v4_rcv
udp_rcv

IP Layer

ip_output

ip_rcv

Datenübertragungsschicht

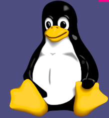
Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Sicherheit

Linux Security Modules – LSM

- Framework für Zugriffskontrollmechanismen durch nachladbare Module
- SELinux von der NSA als Kernel Patch
- Torvalds wollte allgemeines Security-Framework
- Grundstein für die Schaffung von LSM



- Inhalt
- Statistische Daten
- Scheduling
- Memory Management
- Systemaufrufe
- Netzwerk
- Sicherheit

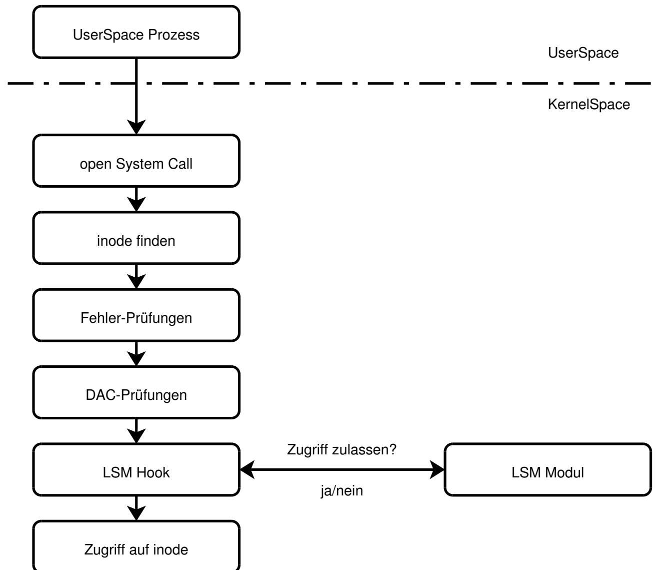
© 2004 Quaritsch Markus, Winkler Thomas

ARCHITEKTUR

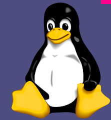
- Zugriff auf Kernel-Objekte regeln
- darf Subjekt (Prozess) auf Objekt (Datei, Socket, ...) zugreifen
- an den entsprechenden Stellen im Kernel Hooks eingebaut
- Secrutiy Module kann Funktionen für die Hooks registrieren welche dann vom Kernel aufgerufen werden
- Hook-Funktionen liefern ja/nein Entscheidung

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas



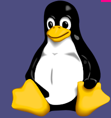
© 2004 Quaritsch Markus, Winkler Thomas



Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

Security Fields

- Möglichkeit sicherheitsrelevante Daten an Kernel-Datenstrukturen anzufügen
- einfache `void*` Pointer
- Locking und Speicherverwaltung bleibt den Modulen überlassen

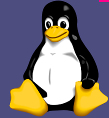


Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

Hooks

- Hook Funktionen eines Moduls in der globalen Datenstruktur `security_ops` vom Typ `struct security_operations` registriert (Liste von Funktions-Pointern)
- diese Funktionen werden von den Hooks in den Kernel-Subsystemen aufgerufen

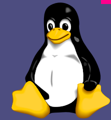


Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas

```
1
2 /* ... */
3
4 struct security_operations {
5
6     /* ... */
7
8     int (*task_setnice) (struct task_struct * p,
9                                     int nice);
10
11    /* ... */
12 };
13
14 /* global variables */
15 extern struct security_operations *security_ops;
16
17 /* ... */
18
19 static inline int security_task_setnice (
20                     struct task_struct *p, int nice)
21 {
22     return security_ops->task_setnice (p, nice);
23 }
```

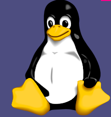
Listing 6: Datenstruktur `security_operations` mit `task_setnice` Funktions-Pointer und zugehöriger `security_task_setnice` Funktion *include/linux/security.h*



© 2004 Quaritsch Markus, Winkler Thomas

```c

/*
 * sys_nice - change the priority of the current
 *                                              process.
 * @increment: priority increment
 *
 * sys_setpriority is a more generic, but much slower
 * function that does similar things.
 */
asmlinkage long sys_nice(int increment)
{
    int retval;

    /* ... */

    retval = security_task_setnice(current, nice);
    if (retval)
        return retval;

    set_user_nice(current, nice);
    return 0;
    }
```

Listing 7: System Call `sys_nice` mit LSM Hook



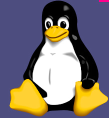
© 2004 Quaritsch Markus, Winkler Thomas

Module Stacking

- Möglichkeit um Module zu kombinieren
- LSM Framework selbst sieht immer nur ein (primäres) Modul
- Stacking basiert auf Kooperation der Module
- Module weiter vorne reichen Hook-Aufrufe durch

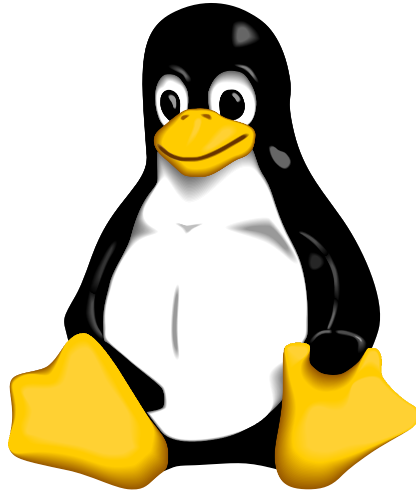
Performance und Einsatzbereiche

- Performance Overhead relativ gering (wenige Prozent)
- bereits mehrere Module verfügbar
- SELinux auf LSM portiert und offizieller Bestandteil von Linux 2.6



Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit

© 2004 Quaritsch Markus, Winkler Thomas



Danke für die Aufmerksamkeit!

© 2004 Quaritsch Markus, Winkler Thomas

Inhalt
Statistische Daten
Scheduling
Memory Management
Systemaufrufe
Netzwerk
Sicherheit