Timing Measurements

Philipp Seidel

22.5.2006

Inhaltsverzeichnis

- viele Computer Aktivitäten sind Zeit gesteuert und laufen meistens vom Benutzer unbemerkt im Hintergrund
- 2 unterschiedliche Typen von Zeitmessung
  - aktuelle Zeit und Datum (Funktionen wie time(), ftime(), gettimeofday())
  - Timer (Funktionen: settimer(), alarm() und delay Funktionen)
- Zeitbestimmung durch einige Hardwareschaltkreisen, Oszillatoren und Zählern erledigt

- Echtzeituhr, unabhängig von der CPU und allen anderen Chips
- CMOS RAM und RTC in separaten Chip (Motorola 146818 oder ähnlicher)
- regelmäßige Interrupts IRQ8 mit Frequenz von 2Hz bis 8,192Hz
- ansprechbar über /dev/rtc; dazu werden die I/O ports 0x70 und 0x71 benutzt
- Admins können RTC lesen und schreiben über Programm clock
  hwclock

- PIN für CLK Signal von externem Oszillographen
- seit Pentium 64-Bit Zähler
- Kernel muss Frequenz des Zeitsignals beachten
- durch höhere Frequenz genauer als PIT
- beim Bootvorgang wird die Frequenz mit der Funktion `calibrate_tsc()` ermittelt

- Interrupt wird mit vom Kernel bestimmt Frequenz aufgerufen
- PIT auch für Tonerzeuger benutzt
- durch 8254 CMOS Chip realisiert, welcher über die I/O Ports Ox40-Ox43 angesprochen wird
- Linux programmiert PIT das er IRQ0 mit Frequenz von ca. 1000 Hz (Zeitintervall Tick genannt, dass Länge wird `tick_nsec` gespeichert)
- bei PCs wird `tick_nsec` auf 999 848 gesetzt (ca. 1000,15 Hz)
- kürzere Ticks einen höher aufgelöster Zeitgeber
- ein paar Macros im Linux Code legen Frequnez der Timer Interrupts fest
- HZ liefert die Anzahl der Timer Interrupts pro Sekunde
- `CLOCK_TICK_RATE` bekommt den Wert 1 193 182 welches die interne Frequenz des Oszillographen des 8254 Chips ist
- LATCH Verhältnis aus `CLOCK_TICK_RATE` und HZ → benutzt um den PIT zu programmieren
- PIT wird durch die Funktion [illegible]() initialisiert

-
  - PIT sehr ähnlich
  - APIC's Timer ist 32 bit
  - erhält Anzahl der Ticks bis Interupt ausgelöst wird (sehr niedrige Frequenzen)
  - abhängig vom Bus Frequenzsignal
-
  - neuer Chip zur Zeitmessung, von Intel und Microsoft entwickelt
  - Chip besitzt 8 unabhängige 32-bit oder 64-bit Zähler
  - jeder Zähler hat eigenes CLK Signal (Frequenz min. 10 MHz)
  - der HPET kann durch Register, welche im Speicher abgebildet werden programmiert werden (ist dem I/O APIC sehr ähnlich)
  - HPET wird PIT in Zukunft ersetzen

- auf nahezu allen ACPI-basierten Motherboards zu finden
- Tackfrequenz fest (ca. 3,58 MHz)
- Zählerstand auslesen über I/O-Port, Adresse vom BIOS während Initalisierungsphase festgelegt
- unabhängig von CPU Frequenz und Spannung

- Einprozessorsysteme: globaler Interrupt (entweder duch den PIT oder den HPET)
- Multiprozessorsystem:
- alle allgemeinen Aktivitäten (z.B. Software Timer) durch globalen Interrupt realisiert
- alle CPU spezifischen Aktivitäten (Zeit eines Prozesses) durch den localen APIC Interrupt realisiert

- es wird eine große Zahl von Datenstrukturen benutzt
- Timer Objekt vom Typ `timer_opts`

`name`	eine Zeichenkette die die Zeit Quelle indentifizie
`mark_offset()`	liefert die exakte Zeit in der ein Zeit Interrupt a
`get_offset()`	Mikrosekunden seit dem letzten Tick
`monotonic_clock()`	Nanosekunden seit der Kernel Initialisation
`delay()`	Wartet eine Anzahl von Loops

- `cur_timer` wird durch `select_timer()` gesetzt undenthält aktuelle Adresse des besten Timer Objekts
- die am meisten verwendeten Timer der 80x86 struktur, in der Reihenfolge, wie sie bevorzugt werden

- `jiffies` Variable speichert Anzahl der Ticks zählt seit Systemstart
- 32-Bit Variable = 50 Tage bis zum Überlauf (Macros `time_after`, `time_after_eq`, `time_before` und `time_before_eq`)
- `jiffies` zur Bootzeit mit 0xfffb6c20 (Wert -300000) initialisiert, Überlauf nach 5 Minuten
- `jiffies_64` ist 64-Bit Variable; in unteren 32-Bit wird `jiffies` abgebildet

- xtime speichert die aktuelle Zeit und das Datum und ist vom Type timespec
- von ihr erhalten die Benutzerprogramme die aktuelle Zeit
- seqlock wird zum sichern einiger kritischer Zeiterhaltungsfunktionen benutzt

- zeitbezogenen Aktivitäten durch PIT gemacht
- Initialisierungsphase `time_init()`
- Initalisieren der `xtimer` Variable
- UNIX Timestamp wird von der RTC gelesen `get_cmos_time()`
- der `tv_nsec` Wert wird so gestzt, dass er genau mit dem bevorstehenden Überlauf von der `jiffies` Variable zusammen fällt
- Initalisieren der `wall_to_monotonic` Variable
- Type: timespec (wie xtime)
- Funktion:
- speichert den Wert von Sekunden und Nanosekunden, die zu xtime dazu gezählt werden müssen, damit ein monotones Hochzählen entsteht ohne Sprünge
- Wenn der Kernel HPET unterstützt; `hpet_enable()`
- `select_timer()`

- `timer_interrupt()` ist die interrupt service routine (ISR) des PIT oder des HPET
- Schützt die Zeitrelevanten Variablen; einführen von `write_seqlock()` auf `xtime_lock` seqlock
- `mark_offset()` vom `cur_timer` Objekt
- `do_timer_interrupt()`
- `xtime_lock seqlock` durch `write_sequnlock()` aufheben
- 1 zurückgeben, damit klar ist, dass der Interrupt ordnungsgemäss behandelt wurde

- können 2 unterschiedliche Zeitgeberquellen benutzen: (PIT oder HPET) und den lokalen CPU Timer
- bei Linux 2.6 wird der PIT oder HPET benutzt um die Software Timer und die Zeit und das Datum aktuell zu halten
- die Prozesssteuerung wird über lokale Zeit Interrupts gemacht
- Globaler Timer Interrupt Handler
- die SMP Version von `timer_interrupt()` unterscheidet sich von der UP Version
- Lokaler Timer Interrupt Handler
- dieser Handler verarbeitet die Zeiterhaltungsfunktioen einer speziellen CPU, genau gesgat Profiling des Kernel Codes und überprüfen, wie lange ein Prozess schon läuft

- Benutzerprogramme erhalten die aktuelle Zeit von der `xtime` Variable
- diese muss periodisch aktualisiert werden
- `update_times()` Funktion
- es wird davon ausgegangen, dass `xtime_lock seqlock` vor dem Funktionsaufruf schon aufgerufen wurde
- die `wall_jiffies` Variable speichert die Zeit seit dem letzten update von `xtime`

- neben wichitgen Zeitfunktionen muss der Kernel auch Statistiken aktualisieren
- überprüfen der CPU Ressourcengrenze des laufenden Prozesses
- aktualisieren der Statistik über die aktuelle workload der CPU
- berechnen der durchschnittlichen CPU Auslastung
- Profiling des Kernel Codes

- `update_process_times()` Funktion
- bei Einprozessursystemen durch den globalen timer Interrupt aufgerufen
- bei Mehrprozessorsystemen durch den lokalen Zeitgeber Interrupt aufgerufen

- berechnen der Durchschnittslast
- Einprozessormaschienen:
- 0 = keine aktiver Prozess (neben dem Leerlaufprozess 0)
- 1 = ein einziger Prozess belastet die CPU
- ¿1 = mehrer Prozesse beanspruchen die CPU
- bei jedem Tick ruft die Funktion `update_times()` die Funktion `calc_load()` Funktion auf welche Prozesse mit Status `TASK_RUNNING` oder `TASK_UNINTERRUPTIBLE` zählt

- Linux besitzt einen minimalistischen Code Profiler
- Profiler verwendet einfachen Monte Carlo Algorithmus
- soll das Profiling eingeschaltet werden muss dem Kernel als Bootparameter `profileN` übergeben werden, wobei 2N die Größe des Codesegments ist
- die Daten können dann aus `/proc/profile` ausgelesen werden
- die Zähler werden zurückgesetzt, wenn man in die Datei schreibt
- Tool: `readprofile`
- Linux 2.6: neues Profiling Verfahren: oprofile benutzbar für Kernel Code, Benutzeranwendungen und Bibliotheken

- bei Mehrprozessorsystemen
- dieses ist nützlich, um kernel Bugs zu finden, die ein Einfrieren des Systems verursachen
- um das Watchdog System zu aktivieren muss der Kernel mit dem `nmi_watchdog` Parameter geladen werden

- Timer werden vom Kernel aber auch von Prozessen benutzt
- viele Gerätetreiber benutzen sie: Diskettentreiber, Parallel Druckertreiber
- Funktionen: `setitimer()` und `alarm()` Systemaufrufe
- 2 Timertypen:
- dynamische Timer, welche vom Kernel benutzt werden
- interval Timer, werden von Prozessen erstellt
- Timer nicht für Realtime Anwendungen geeignet

- können dynamisch erstellt und gelöscht werden
- keine Begrenzung
- sie werden in der `timer_list` Struktur gespeichert

nanosleep() Systemaufruf

- Routine ist `sys_nanosleep()`, welche als Parameter dein Zeiger auf eine `timespec` Struktur übergeben bekommt
- unterbricht Prozess für angegebenes Zeitintervall

- nix

- time()
- gibt den UNIX Timestamp zurück
- gettimeofday()
- gibt eine Data Struktur zurück, die den UNIX Timestamp beinhaltet und die Anzahl der Microsekunden, die in der letzten Sekunde vergangen sind
- eine zweite Datenstruktur timezone wird im Moment nicht benutzt
- ftime() ist nicht mehr als Systemaufruf vorhanden, sie hat den UNIX Timestamp zurück gegeben und die Anzahl der Millisekunden

- wird zum Korrigieren der Systemzeit benutzt, da die Hardware Uhr manchmal etwas abdriftet
- die Funktion erhält als Parameter einen Zeiger auf eine timex Struktur und updatet die Kernel parameter
- mit der selben Struktur werden die aktuellen Kernel Werte zurückgegeben, die Werte werden dann z.B.: von `update_wall_time_one_tick()` verwendet um Anzahl der Microsekunden zu korrigieren, werden bei jedem Tick zum Wert von `xtime.tv_usec` hinzugezählt

- Linux erlaubt es User Mode Prozessen interval timer anzulegen, diese Haben nichts dem Intervall Timer Chip zu tun
- Timer erzeugen Benutzersignale welche periodisch an einen Prozess geschickt werden
- Timer auch als Älarmfunktionv̈erwendbar
- Der Timer wird durch den POSIX Systemaufruf `setitimer()` aktiviert: `ITIMER_REAL`, `ITIMER_VIRTUAL`, `ITIMER_PROF`