Eksamen 2012  
3. semester  
I3MPS - Microprocessor systemer  
Kasper Nissen

**Spørgsmål 7**

**IOCTL + Kernel Timers**

**Spørgsmål**

* Hvad benytter man IOCTL kald til?
* Hvordan implementeres ioctl funktioner i en device driver og hvordan tilgås disse?
* Hvordan benytter man timers / delays i kernen?
* Hvilke fordele / ulemper har de forskellige delay / timer typer?

**Pensum:**

* Linux Device Driver 135-147 (IOCTL)
* Linux Device Driver 183-202 (Timers/Delay)

**Exercise 8**

**IOCTL – Hvad og hvorfor?**

De flester drivere ønsker at kunne tilføje ekstra funktionalitet end blot read og write metoder. Muligheden for at tilføje ekstra hardware kontrol via en device driver som f.eks. opsætning af en baud rate, eller skub et medie ud – det er her IOCTL kommer ind i billedet.

I user space har ioctl kaldet følgende funktions prototype:  
  


Som det ses så er denne prototype lidt speciel, da det tredje argument blot er … - dette skyldes at det tredje argument kan variere i type alt efter hvilken form for kontrol som ioctl kaldet skal kunne levere.

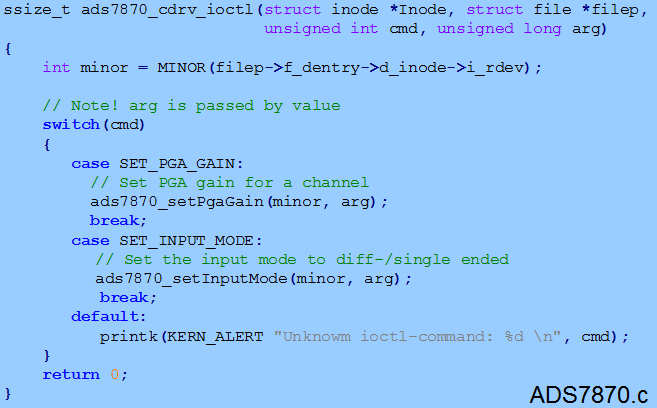
I Kernelspace ser prototypen lidt anderledes ud:

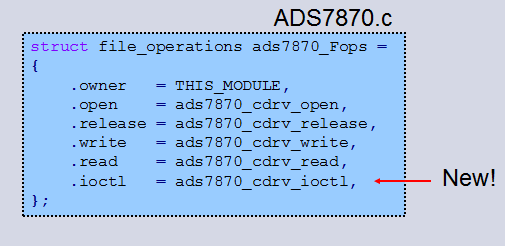


Det er selvfølgelig denne type der skal implementeres i vores driver, hvor i mod den foregående prototype kaldes fra et program i user space.

**Hvordan implementeres ioctl funktioner i en device driver og hvordan tilgås disse?**

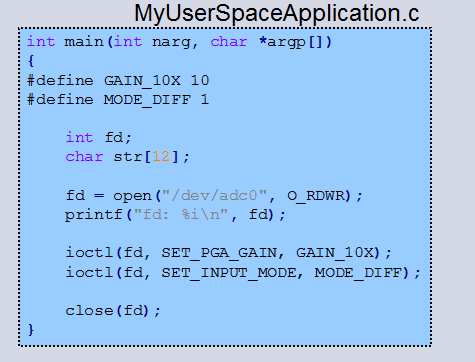
De 2 første pointere i prototypen, dækker over fd(file descriptoren) i userspace. Argumentet cmd bruges til at switche mellem de forskellige funktioner som ioctl kaldet skal kunne udføre. Selve implementeringen af dette kan ses i kode eksemplet nedenfor:



Som det ses i eksemplet, så kan vi altså swicthe på cmd, og få udført nogle forskellige funktioner. Cmd i prototypen til vores driver svarer til cmd i prototypen til user space.

I alle device drivere har vi et sted hvor vi har lavet en struktur over de samlede filoperationer, her skal selvfølgelig også tilføjes et ioctl kald.

Eksempel på userspace applikation:



Som det ses ovenfor åbnes det indsatte kernemodul. Dernæst kaldes ioctlr med nogle forskellige parametre.

**Andet IOCTL-LIR**

* IOCTL kommandoer er globale
* Disse kald tildeles unikke numre -> tjek ioctl-number.txt
* Hver gang man foretager et ioctl-kald med nyt argument, skal der gives et nyt nummer.
* Gør brug af makroer(define)
* Implementering:
  + Kerne:
    - IOCTL kald implementeres med switch-case der tjekker på det unikke nummer og udfører operationen i kernen
  + Applikation:
    - Herfra kaldes ioctl med en filedescriptor, kommandoen (unikke nummer) og et argument
    - Man kan give argumentet på forskellige måder:
      * set by value
      * set by pointer
      * get by value
      * get by return

**Opsamling på IOCTL**

* En IOCTL kommando er global i hele kernen
* Er repræsenteret ved et specifikt nummer
* Så snart modulet er indsat i kernen er ioctl nummeret kendt
* Der kan kun sendes et argument med (kan dog være en pointer)

**Timers/delays**

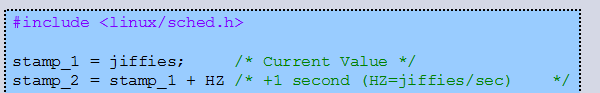
**Der er forskellige mekanismer til timing og delays:**

Linux bruger en system timer interrupt til at holde styr på tiden, denne timers resolution svinger fra mellem 50-1000 ticks pr. sekund alt efter hvilken hardware systemet kører på. Ved boot time blive en værdi indlæst i HZ som er defineret i <linux/param.h> initieret.

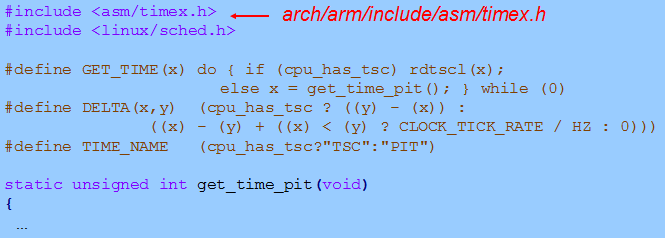
Hvergang et time interrupt sker, tælles værdien af den interne counter op. Denne counter bliver fra boot inititeret med værdien 0. Denne counter er en 64-bit counter og kaldes for jiffies.Denne timer ligger i biblioteket <linux/jiffies.h> men bliver også inkluderet ved <linux/sched.h>. Jiffies er read only.

Jiffies counter flipper rundt ved 2^64 timer ticks.

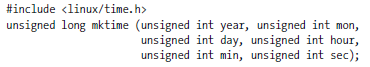
Eksempel:



Der findes også processer specifikke timere, hvis man ønsker mere præcision, men en sådan er selvfølgelig CPU specifikke. Et eksempel på dette:



Der findes ligeledes en metode som returnere den aktuelle tid i minutter, timer, dage osv.. Men i realiteten vil dette aldrig være nødvendigt i et kernemodul. Der findes bedre metoder i user space der klare denne opgave. Og der er i realiteten ikke rigtig noget tidspunkt hvor det er nødvendig.



Dog kan der forekomme steder hvor et absolut timestamp kan være nødvendigt, og denne prototype er som følger:



**Delays**

Device drivere har ofte brug for at delaye udførslen af noget kode for et stykke tid, f.eks. indtil hardwaren har udført en opgave.

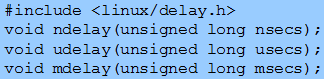
Busy waiting

Hvis man ønsker at udskyde en execution i mange clok ticks, kan det gøres ved en simpel busy waiting, som dog ikke anbefales. Som kan se sådan ud:



Dette skader systemets performance, da loopet vil låse processoren og der vil ikke være mulighed for andre tråde eller processer at komme til.

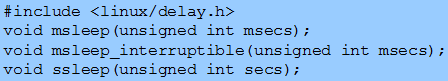
Der findes funktioner som kan bruges hvis der ønskes busy-waiting, dog skal det være meget små busy-waits, for ikke at stoppe alting, jf. ovenfor.



Hvis der ønskes brug af interrupts, kan følgende metode f.eks. bruges, til at sætte et specifikt punkt for timeout.



Hvis vi i stedet ønsker at ligge CPU’en til at sove kan vi også bruge følgende funktioner til små delays:



Hvis vi blot ønsker at udskyde udførslen af noget kode kan vi bruge:

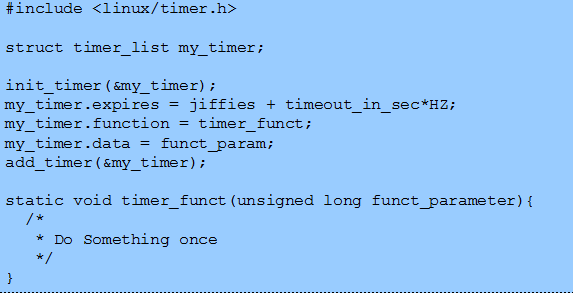


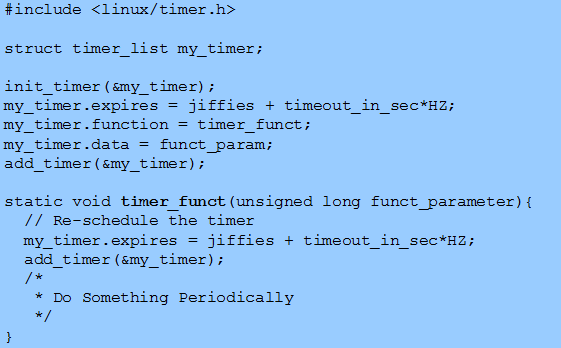
Dette vil være langt bedre, da vi ikke sætter CPU’en til at sove, men lader scheduleren styrer det.

**Kernel timers:**

* Brug en timer hvis du ønsker at udsætte udførslen af en process
* Kerne timere fungere ved brug af software interrupts
* Timer interrupts er asynkrone
* Implementeringen minder om et interrupt
  + Request
  + Service Routine
* Da timere er opbygget omkring interrupts, kan disse IKKE bruges i user space

**Eksempel på One-shot timer:**



**Periodisk timer:**

**Opsamling på timers og delays.**

* **Jiffies:**
  + Global variabel
  + Bliver nulstillet ved system boot
  + Bliver inkrementeret vha. et timerinterrupt
  + Ved at læse jiffies 2 gange med et bestemt mellemrum, kan man angive tidsforskellen.
  + Derefter sammenlignes værdien med den globale konstant HZ der angiver værdien for 1 sekund
  + 50-1000 ticks
* **Timestamp counter:**
  + En 64-bit variabel
  + Inkrementeres ved hver clockcycle
  + Høj opløsning -> Løber rundt på 4,2 sekunder ved 1 Ghz
  + Kan aflæses fra både userspace og kernelspace.
* **Busy-waiting:**
  + Laves vha. while-loop
  + Tjekker på jiffies og en forudsat jiffies-værdi
* **Korte delays:**
  + Busy-wait funktioner (mdelay, udelay, ndelay)
  + sleep
* **Kernel Timers:**
  + Skedulerer en action til at ske senere uden at blokere nuværende process
  + Bliver brugt