METODER OG FUNKTIONER

Funktioner:

Mdelay()

mdelay() — TL;DR (på dansk)

- Ö Forsinker udførelsen i millisekunder
- Busy-wait (CPU'en er aktiv hele tiden)
- Mere præcis end msleep()
- X Bruger flere CPU-ressourcer
- X Ikke strømbesparende
- ⚠ Brug kun ved behov for høj præcision, fx tidlig boot eller hardware-nær kode
- VS Brug msleep() når:
- Du kan leve med lidt unøjagtighed
- · Du vil lade CPU'en arbejde videre med andre opgaver

MSleep()

msleep() — TL;DR (på dansk)

- Sover i millisekunder
- X Ikke super præcis (kan variere lidt afhængigt af systemets "jiffy"-tid)
- God til energibesparelse og multitasking
- Afhænger af scheduler
- Brug mdelay() kun hvis du har brug for høj præcision og kan acceptere høj CPU-brug

Alloc_chrdev_region()

Forklaring:

- alloc_chrdev_region()
 - Denne funktion allokerer et major- og minor-nummer i kernelens device-tabeller.
 - Det er første skridt for at kunne knytte en character device til kernelens device-node system.
 - Den returnerer et dev_t, som bruges videre i cdev_add() og device_create().

Device_create()

- device_create()
 - Opretter en device-fil under /dev , men kræver at du allerede har major/minor via alloc_chrdev_region() .

Cdev_add()

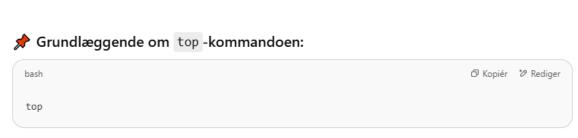
- - Registrerer en character device i kernelens cdev-system, men bruger dev_t som allerede er allokeret.

Class_create()

- X class_create()
 - Bruger man til at oprette en klasse i /sys/class , typisk før device_create() .

TOP

top er en kommandolinje-baseret overvågningskommando i Linux, der viser realtidsinformation om systemets ressourcer og processer.

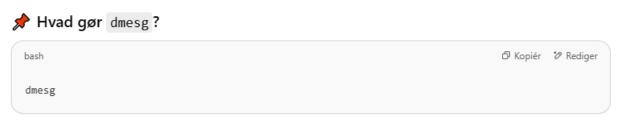


Dette viser en dynamisk og løbende opdateret oversigt over:

- CPU-brug
- Hukommelsesforbrug
- Kørende processer
- Systemets belastning (load average)
- Brugere og systemtid

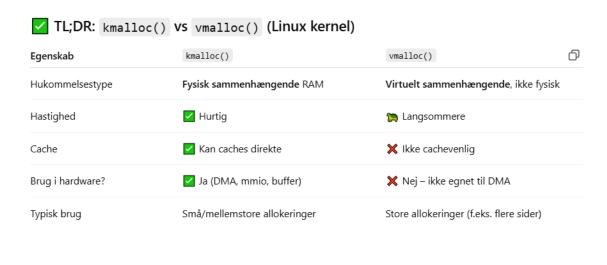
DMESG

Kommandoen dmesg i Linux viser **kernelmeddelelser** – altså beskeder fra Linux-kernen om systemets hardware og drivere.



Den viser output fra **ring buffer** (en midlertidig log), som kernen skriver til under systemets opstart og ved hardwarebegivenheder.

VMALLOC OG KMALLOC'



UDELAY()

- ✓ TL;DR: udelay()
 - udelay(n) laver en aktiv forsinkelse i n mikrosekunder (µs)
- Bruges i Linux kernel code (ikke i user-space!)
- Eksempel:

Bruges typisk til:

- Tidskritisk hardware-initialisering
- Vent på kort hardware-respons (f.eks. GPIO, SPI)

Bemærk:

- Blokerer CPU'en aktivt → ineffektiv ved længere ventetid
- Til længere delays: brug mdelay() (millisekunder) eller sleep-funktioner

Nøgleidé:

udelay(n) = vent præcist n μ s, men spild CPU-tid mens du gør det.

Konklusion:

- udelay() blokerer CPU aktivt, ja
- Men den blokkerer ikke i kernel-scheduling-forstand, og kan derfor bruges i interrupts

Question 2 (5 points) Saved		
I	hvilke situationer vil man vælge at bruge udelay frem for usleep? (Vælg 2)	
S	elect 2 correct answer(s)	
	Når der er behov for at frigive CPU ressourcer i systemet	
(✓ Når præcision er vigtigt	
	Når præcision ikke er afgørende	
	Når forsinkelsen er større end 5 ms	
(✓ Når man ikke må blokere, f.eks. i et interrupt	

Her er der flere lignende:

Herunder:

Udelay()

Mdelay()

Usleep_range()

Msleep()

Schedule_timeout()

Msleep_interuptible()

🖒 Delay- og sleep-funktioner i kernel space

Funktion	Enhed	Туре	Må bruges i interrupt?	Præcision	Kommentar
udelay(n)	Mikrosekunder	Busy-wait	✓ Ja	✓ Høj	Aktiv ventetid (CPU spildes)
mdelay(n)	Millisekunder	Busy-wait	✓ Ja	▲ OK, men ineffektiv	Bruges sjældent – loopet udelay
<pre>usleep_range(min, max)</pre>	Mikrosekunder	Sleep (kontekstskift)	🗶 Nej	✓ Nogenlunde	Bedste præcise sleep i ikke- interrupt
msleep(n)	Millisekunder	Sleep	X Nej	X Lav	Simpel sleep, lav præcision
<pre>schedule_timeout()</pre>	Jiffies	Sleep + Timeout	🗶 Nej	▲ Afhænger af HZ	Bruges til vent med timeout i tasks
msleep_interruptible()	Millisekunder	Sleep (kan afbrydes)	X Nej	🗶 Lav	Kan afbrydes via signaler

Vigtige tommelfingerregler:

- Interrupt context = udelay()
- <10ms delay og ikke i interrupt = usleep_range()</p>
- > >10ms delay = msleep() eller schedule_timeout()

OPEN()

- open() TL;DR
- Kaldes, når en bruger åbner enheden, f.eks. via open("/dev/mit_device", ...).
- Er en del af file_operations structen:

```
c
struct file_operations fops = {
    .open = my_driver_open,
    ...
};
```

Formål

- Bruges til at initialisere enhedstilstand, f.eks.:
 - · Allokere ressourcer (f.eks. buffer eller IRQ)
 - · Tjekke hardwarestatus
 - · Forberede enheden til brug

⚠ Bemærk

- Den må ikke sove hvis kaldt fra kernelkontekst (sjældent tilfældet i open).
- Hvis du requester en interrupt her, skal du huske at frigive den i release().

Brug open() til:

- Starte hardware
- Låse ressourcer
- Tilknytte private_data til filp->private_data

RELEASE()

release() - TL;DR

- Kaldes, når en bruger lukker enheden, f.eks. via close(fd);
- Modsvarer open() det er her, du **rydder op** og frigiver ressourcer.

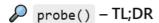
Typisk brug

- Frigive:
 - IRQ'er (hvis du brugte request_irq() i open())
 - · Dynamisk allokeret hukommelse
 - Hardware-ressourcer
- · Lukke kommunikation med hardware, hvis nødvendigt

→ Brug release() til:

Formål	Eksempel
Frigive IRQ	<pre>free_irq()</pre>
Rydde buffer/hukommelse	<pre>kfree(filp->private_data)</pre>
Lukke forbindelse	Afslutte samtale med hardware

PROBE()



- probe() kaldes automatisk, når en driver matches med en hardware-enhed (typisk via device tree eller platform bus).
- Det er stedet, hvor du initialiserer enheden, f.eks.:
 - · Allokerer ressourcer (I/O memory, IRQs, buffers)
 - Opretter device filer (via device_create())
 - · Registrerer enhedens tilstand

Hvorfor bruge probe()?



INIT()

init() - TL;DR

- init() er ikke en del af file_operations , men er kernel-modulets initialiseringsfunktion.
- Det er den funktion, som kaldes én gang, når modulet loades med insmod.

Typisk brug:

- Registrere driveren i systemet
- Oprette device class
- · Allocere major/minor nummer
- Registrere file_operations (f.eks. via cdev_add)

TL;DR konklusion:

init() gør dette:

Registrerer driver (cdev, class, device)

Allokerer systemressourcer (major/minor)

Kaldes én gang ved modulindlæsning

EXIT()

exit() - TL;DR

- exit() er afslutningsfunktionen for et kernel modul.
- Kaldes automatisk, når du fjerner modulet med rmmod.
- Bruges til at rydde op efter init(), så der ikke lækker ressourcei

Typisk oprydning i exit():

- Fjern /dev/ -filen: device_destroy(...)
- Slet klassen: class_destroy(...)
- Fjern device fra kernel: cdev_del(...)
- Frigiv device number: unregister_chrdev_region(...)

w Hush.

Alt du allokerer i init(), skal frigives i exit() – ellers lækker du ressourcer (filer, klasser, IRQs osv.).

TL;DR Konklusion

exit() gør dette: Kaldes ved rmmod Rydder op efter init()

Frigiver enhedsnumre, sletter device/class/IRQ

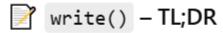
READ()

- read() TL;DR
- read() kaldes, når brugeren forsøger at læse fra en device, f.eks.:

TL;DR Konklusion

read() bruges til	
Returnere data fra driver til bruger	
Kontrollere hvor meget data der gives	
Sikre korrekt kopiering via copy to user	

WRITE()



write() kaldes, når en bruger skriver til din device, f.eks.:

TL;DR Konklusion

write() bruges til...

Modtage data fra bruger til kernel-driveren

Gemme, tolke eller videresende det til hardware

Logge eller validere brugerinput i kernel space

ENHEDSREGISTRERING

Alloc_chrdev_region

Register_chrdev

Cdev_init

Cdev_add

Cdev_del

Unregister_chrdev_region



Funktion	Beskrivelse
<pre>alloc_chrdev_region()</pre>	Allokerer major/minor nummer
<pre>register_chrdev()</pre>	Simpel device-registrering (ældres metode)
<pre>cdev_init()</pre>	Initialiserer cdev struct
cdev_add()	Registrerer character device
cdev_del()	Afregistrerer device igen
<pre>unregister_chrdev_region()</pre>	Frigiver major/minor nummer

PLATFORM/DEVICE-DRIVER INTEGRATION

Of_match_table

Platform_get_resource

Devm_*

Platform/device-driver integration

Funktion	Beskrivelse
<pre>probe()</pre>	Kaldes når enheden matches
remove()	Kaldes når enheden fjernes
of_match_table	Match af device tree noder
platform_get_resource()	Få adgang til I/O eller IRQ
devm_* -funktioner	Automatisk hukommelseshåndtering

DIVERSE NYTTGE FUNKTIONER:

Copy_to_user

Copy_from_user

Kzalloc og kmalloc

Kfree

Printk

Wake_interuptible

Wait_event_interuptible

Diverse nyttige funktioner

Funktion	Beskrivelse
<pre>copy_to_user()</pre>	Kopiér data til user-space
<pre>copy_from_user()</pre>	Kopiér data fra user-space
<pre>kzalloc() / kmalloc()</pre>	Allokerer kernel-hukommelse
kfree()	Frigør hukommelse
<pre>printk()</pre>	Logger til kernel log (dmesg)
<pre>wake_up_interruptible()</pre>	Vækker tråd i ventetilstand
<pre>wait_event_interruptible()</pre>	Blokering med wakeup

MAJOR OG MINER FUNKTIONER

Funktion	Formål
MKDEV()	Kombinér major + minor til dev_t
MAJOR()	Udpak major fra dev_t
MINOR()	Udpak minor fra dev_t

F_pos			
F_inode			
F_path			
F_owner			
<pre>f_pos</pre>			
Filens læse/skriveposition (offset).			
Bruges i f.eks. read() og write() til at holde styr på hvor i filen man er.			
<pre>f_inode</pre>			
Pointer til inode-strukturen, som beskriver selve filen på disk.			
Bruges til at få adgang til metadata som filtype, størrelse, device m.m.			
♦ f_path			
Indeholder sti og mount-info for filen.			
Giver adgang til filens placering i filsystemet (dentry , mnt).			
← f_owner			
Bruges til asynkron I/O og signalering (f.eks. SIGIO).			
Indeholder info om hvilken proces der ejer filen ift. signalhåndtering.			

 (\downarrow)

REGISTER_CHRDEV

TL;DR:

register_chrdev() is a Linux kernel function used to register a simple character device driver.

- Inputs:
 - major : major number (use 0 to auto-assign)
 - name : device name
 - fops: file operations (e.g., read, write)
- Returns: major number or error code
- Used for: creating basic character devices
- **Note:** It's outdated prefer alloc_chrdev_region(), cdev_init(), and cdev_add() for modern drivers.