Principper for Samtidighed og Styresystemer

Drivere

René Rydhof Hansen

April 2008

Ekstraforelæsning om memory management

Tirsdag den 20. maj kl. 10–12 i auditorium 0.1.95 (formentlig).

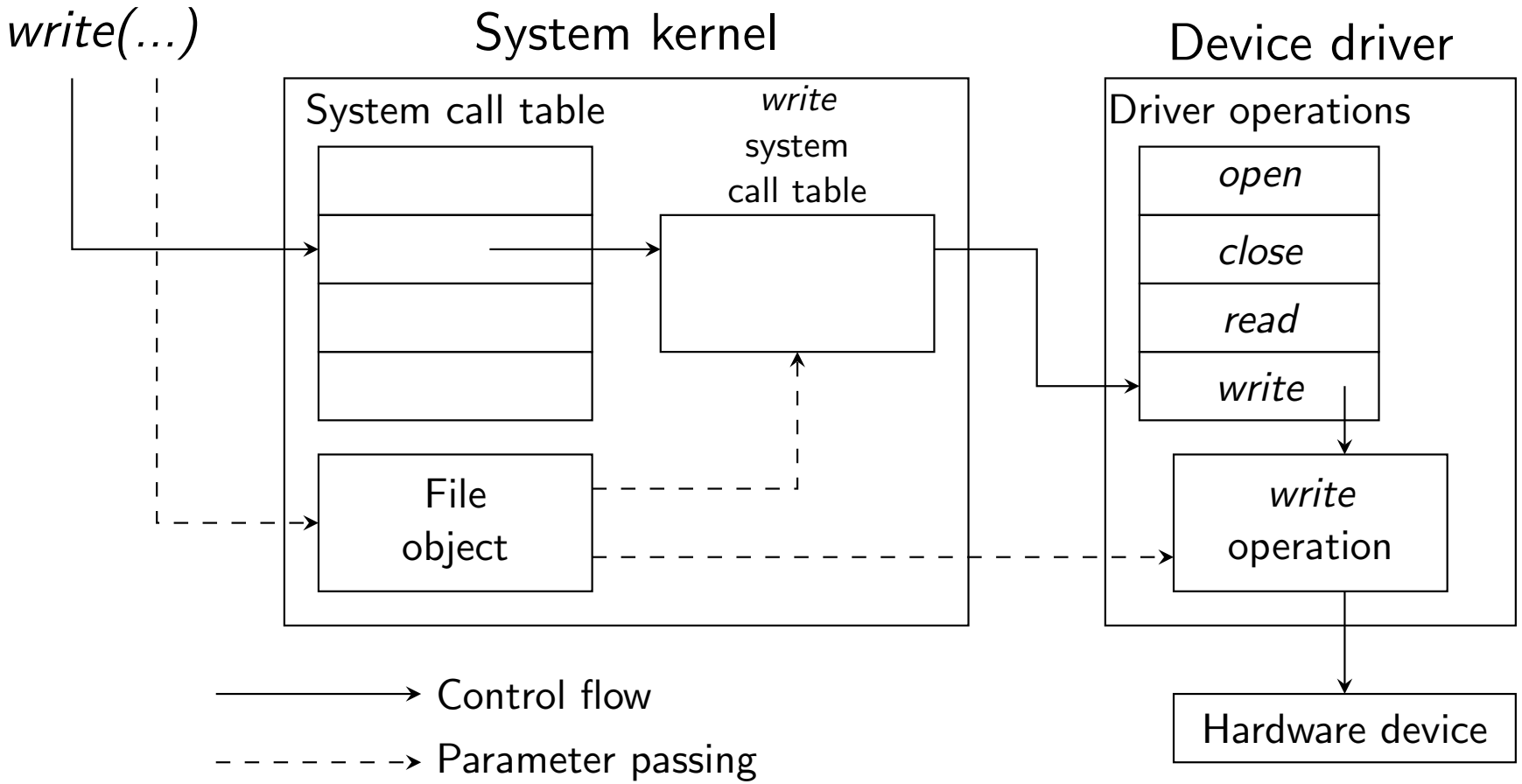
Opgaver

- Opgave 1: Memory management
- Opgave 2: Processorspecifikke mutex-instruktioner
- Opgave 3: Processorspecifikke priviligerede instruktioner
- Opgave 4: Processorspecifik exception-håndtering
- Opgave 5: Buddy blocks

Mål

- At kunne redegøre for hvad en "device driver" er og hvad den bruges til
- At kunne redegøre for grænsefladerne mellem kernen og device drivere
- At kunne definere og redegøre for de forskellige type devices
- At kunne implementere simple device drivere (Linux)
- At kunne redegøre for I/O schedulering
  - FIFO, CBF, SCAN, C-SCAN

Eksempel



- File object peger på rette device driver
- Hvad med den kaldende tråd?

Oversigt

- Hvad er drivere?
- Grænseflader
- Device drivere
- I/O Manager
- Driverstruktur
- Filsystemdrivere

Hvad er en driver?

- Et "modul" til styresystemet
- Styresystemets grænseflade til hardware
  - Diske: floppy, optiske, etc.
  - Keyboards, mus
  - USB, firewire, serielle porte, printerporte, bluetooth
  - Netkort
- Virtuelle enheder
  - VPN
- Plugins
  - Filsystemer
  - Schedulering (policy)
  - Filtrering (firewall)

En driver er et program

- Ligger oftest i “shared kernel space”
- Kører i kernel-mode
  - Adgang til hardware
  - Effektivitet
  - Ingen programbiblioteker
  - Ingen garanti om kontekst
  - Ingen hukommelsesbeskyttelse
  - Skal kunne køre på en multiprocessor-maskine
  - Skal kunne håndtere låse og låsning korrekt
    - Deadlocks i drivere
    - Microsoft SLAM model checker
- Skal, generelt, være fejlfri
- Drivere kan være den eneste mulighed for at få brugerkode kørt i kernel mode

Drivere og OOP

- (Linux-)Drivere skrives altid i C
- Programmeret i en objekt-orienteret stil

Grænseflader

- Kernens grænseflade mod driveren
  - Eksempel: hukommelsesallokering

```
void  *kmalloc(sice_t size, int flags)
void  printk(...)
```

- Driverens grænseflade mod kernen
  - Registrering og afregistrering af drivere

```
int  init_module()
void cleanup_module()
```

  - Tænk: Java/C# interfaces
- Referencetæller
  - `MOD_INC_USE_COUNT`, `MOD_DEC_USE_COUNT`

Grænseflader

- Windows Driver Model (WDM)
  - Standardiseret driverinterface
    - Sikrer at samme driver kan bruges på tværs af Windows versioner
  - Kræver standardiseret API og ABI
    - Application Programming Interface (API)
    - Application Binary Interface (ABI)
- Linux
  - Stabilt API indenfor en stabil kerneserie
  - Stabilt ABI er ikke defineret

Drivereksempel: Hello World!

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

static int  init_module()
{
  printk(KERN_ALERT, "Hello World!");
  return 0;
}

static void  cleanup_module()
{
  printk(KERN_ALERT, "Goodbye");
}

MODULE_LICENSE("GPL");
MODULE_AUTHOR("L.Torvalds");
```

Grænseflade mod user-space

- Systemkald
  - Kræver tildeling af et systemkaldsnummer
  - Frarådes p.gr.a. kobling med systembiblioteket
- Device files `/dev`
  - Block og character devices
  - Major nummer identificerer driveren
  - Minor nummer identificerer enheden
  - Problemer med tildeling af major nummer
    - `mknod`, `devfs`, `udev`
- `Proc`-filsystemet `/proc`
  - Driveren registrerer "filer" i `proc`-filsystemet
- Eget filsystem
  - Driveren eksponerer et helt filsystem

Device drivere: Typer

- Character devices
  - Strøm af tegn
- Block devices
  - Random access
  - Data læses i blokke
- Netværks-drivere
  - Næsten som blok devices, men skal behandles separat

Character devices

- En strøm af tegn
  - Keyboard, mus, konsol, serielle porte
  - `/dev/null`, `/dev/zero`, `/dev/random`
- Grænseflade (interface)
  - Filoperationer
    - `seek`, `read`, `write`, `readdir`, `select`, `ioctl`, `mmap`, `open`, `flush`, `close`, ...
  - Registreres ad modulet
    - `module_register_chrdev`
    - `module_unregister_chrdev`
- Eksempel: `/dev/zero`
  - Leverer en strøm af 0'er
  - `dd if=/dev/zero of=/home/foo/overwrite.me count=512`

Eksempel: Zero device

```
int device_open(inode *i, file *f) {
 MOD_INC_USE_COUNT;
 return 0;
}
int device_close(inode *i, file *f) {
 MOD_DEC_USE_COUNT;
 return 0;
}
ssize_t device_read(
   file *f, char *buffer,
   size_t length, loff_t *offset) {
 int i;
 for(i = 0; i < length; i++)
   buffer[i] = 0;
 return length;
}
ssize_t device_write(
   file *f, const char *buffer,
   size_t length, loff_t *offset) {
 return length;
}
```

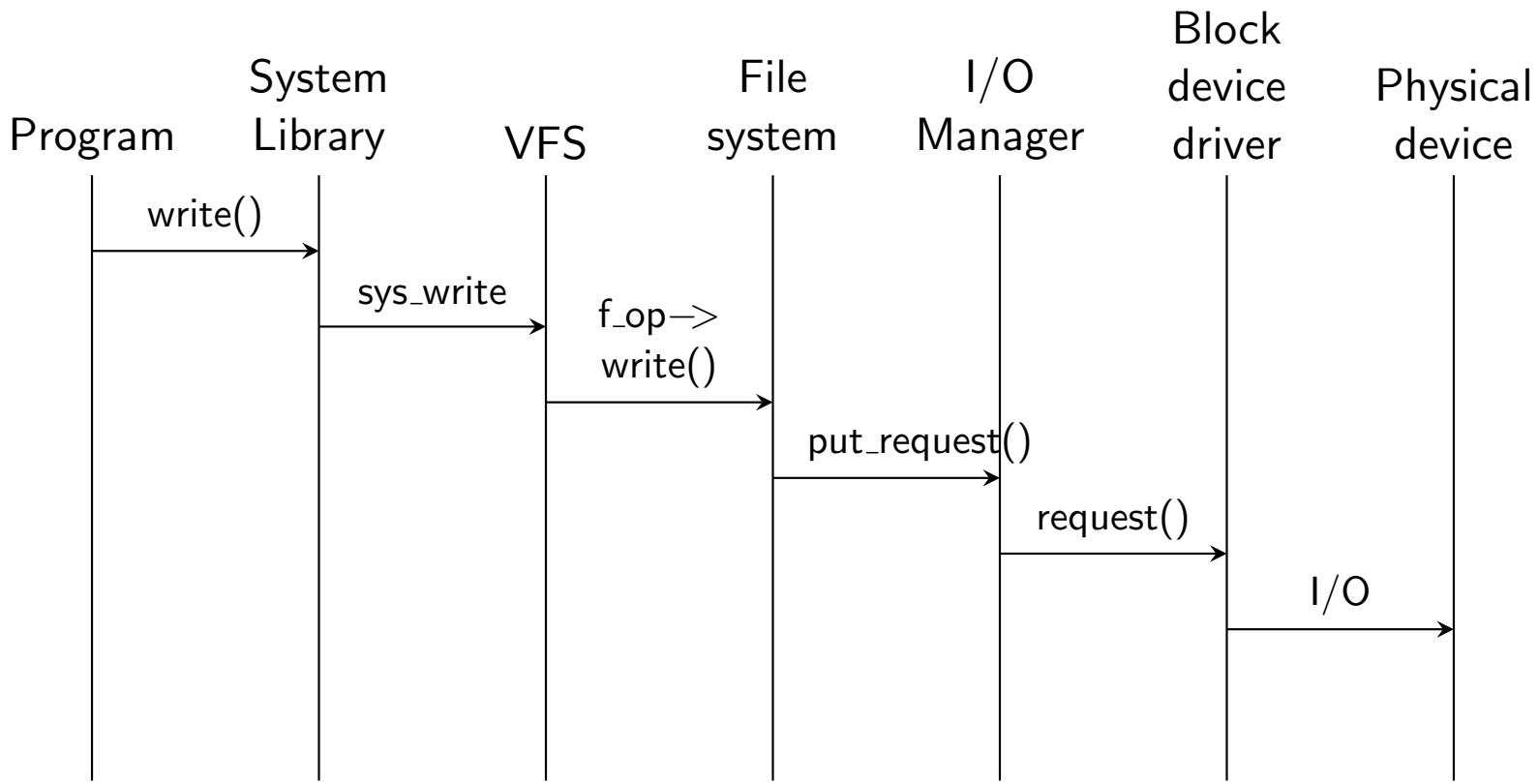
```
file_operations fops = {
 NULL,        /* seek */
 device_read, device_write,
 NULL /* readdir */, NULL /* select */,
 NULL /* ioctl */, NULL /* mmap */,
 device_open,
 NULL,        /* flush */
 device_close
};

int major;

int init_module() {
 major = module_register_chrdev(0,"null dev
 if(major < 0) { printk("error"); return maj
 return 0;
}
void cleanup_module() {
 int n = module_unregister_chrdev(major,"nu
 if(n < 0) printk("error");
 return 0;
}
```

Block devices

- Data overføres i et antal blokke af fast størrelse
- Ingen ordning (random access)
  - Diske: hard, floppy, optiske, ...
- Formateres oftest med et filsystem



I/O Manager

- Buffer cache
  - Cacher ofte læste data
  - Forsinket skrivning (lazy writing)
- I/O Schedulering
  - Optimer global datagennemstrømning (data flow)
  - Fairness
- Datagennemstrømning
  - Sammensmeltning
  - Sortering
- Fairness
  - Skrivning er asynkron af natur
  - Læsning er synkron af natur
  - Latenstider for læsning er vigtigere end for skrivning

I/O Schedulering

- FIFO
  - First come first served
  - Fair men dårlig ydelse
- Closest block first
  - God ydelse, men ikke fair
- SCAN
  - Elevatoralgoritme (ikke altid god for elevatorer)
  - Fastholder retning for læsehovedet
  - Vend retning ved yderste/inderste cylinder
  - Nogenlunde fair, nogenlunde ydelse
- C-SCAN (cyklisk scan)
  - Som SCAN, men vender ikke retning (springer tilbage)
  - Lavere latenstider end SCAN

I/O Schedulering

Example

Request sequence: 20, 5, 23, 16, 27, 63, 28

- FIFO
- Shortest seek first
- Elevator algorithm (SCAN)
- C-SCAN

Problemer med den naive elevatoralgoritme

- Problem: Skrivning udsulter læsning

```
for(...) {
  buf = buf +
    read(file,buf,4096);
}
```

```
for(...) {
   buf = buf +
     write(file,buf,4096);
}
```

- Løsning: Giv læsning højere prioritet
  - Deadline I/O scheduler i Linux gør dette
- Problem: Læsning ødelægger ydelse
  - Søgetiden for læsning er dyr
- Løsning: Holde en lille pause efter læsning
  - Læseren kan nå at afgive næste request
  - Anticipatory I/O scheduler i Linux gør dette

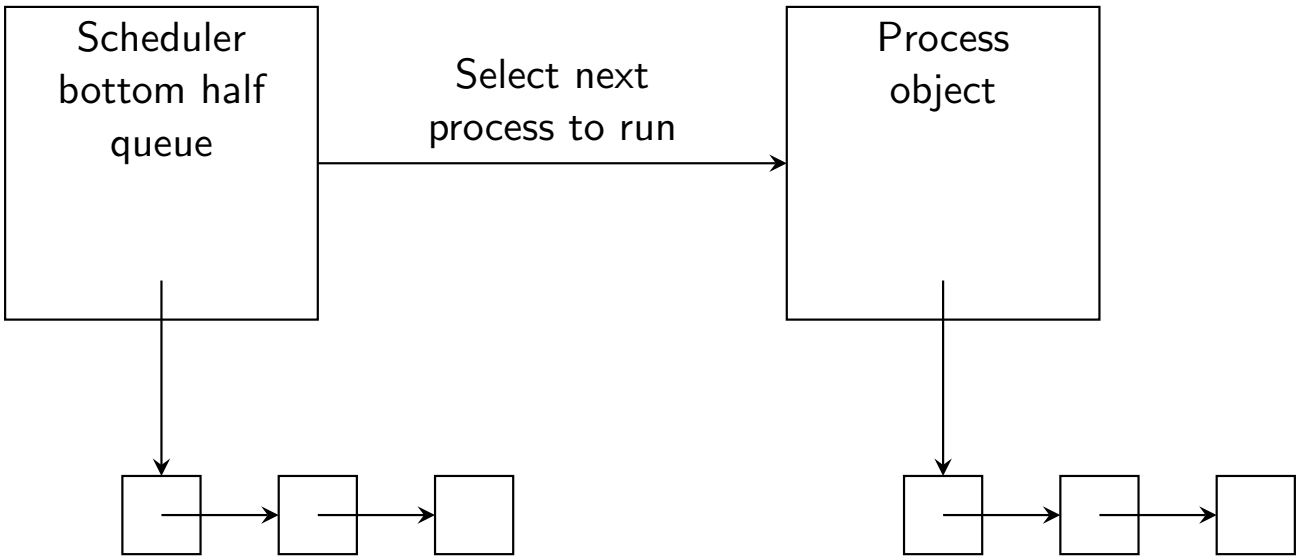
Interruptbehandling

- Interrupt-baseret I/O er normen
  - Drivere kan registrere egen ISR
- ISR udføres i speciel interrupt context
  - Interrupts er uforudsigelige
  - Ingen antagelser om adresserum
  - Egen stak eller bruger af den afbrudte proces' stak
- Hastigheden for ISR er afgørende
  - Mistede interrupts
  - Responstid
  - Blokerende kald
  - User-space hukommelse

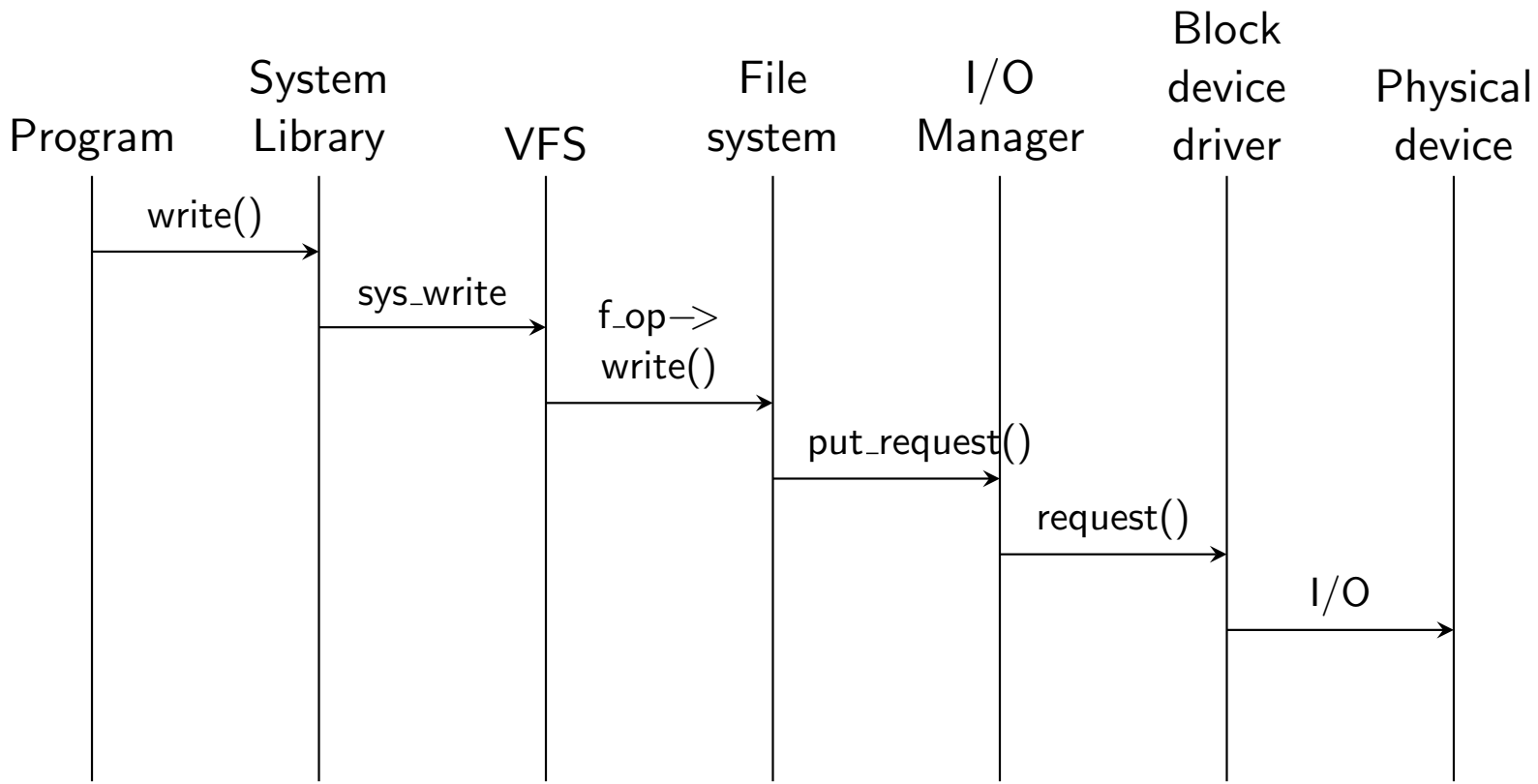
Interruptbehandling

- Top half
  - ISR
  - Foretager kun essentielle ting
    - Fx bekræfte interrupt
  - Schedulerer bottom half
- Bottom half
  - Interrupt context
    - Kræver ikke særlig processkontekst
    - Deferred Procedure Call på Windows
    - Tilføjes scheduler kø
  - Process context
    - Kræver særlig processkontekst
    - Asynchronous Procedure Call på Windows
    - Tilføjes processpecifik scheduler kø
  - Udføres af scheduleren
  - Må blokere

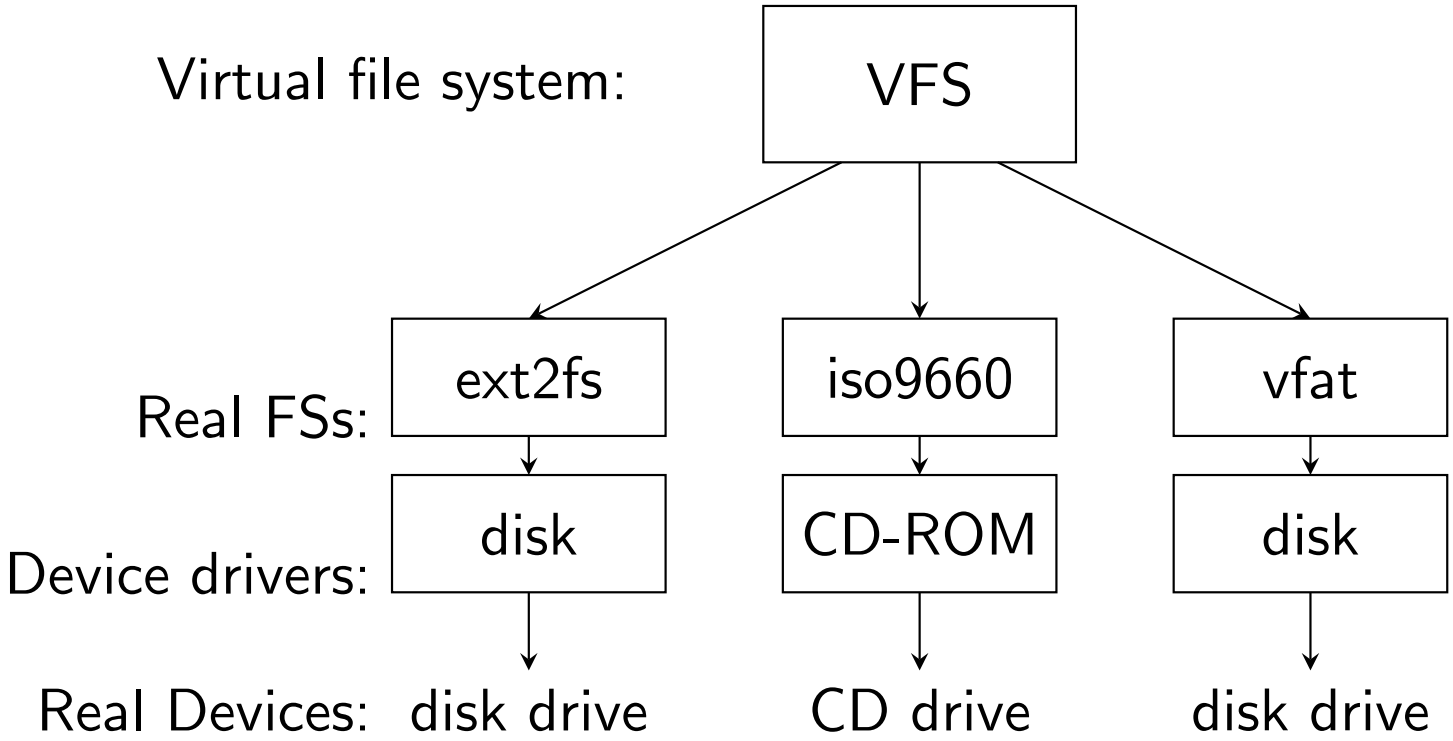
Bottom half eksekvering



Eksempel: Virtual File System



Eksempel: Virtual File System



- `open("/mnt/cdrom/dir/foo",...)`

Opsummering og næste gang

- Drivere er kerne moduler
- Device drivere indkapsler device-specifik kode
- Interruptbehandling er delt i to
- VFS
- I/O Manager
- Næste gang: