LÄNKAR TILL FILER

- Man kan skapa flera pekare till samma fil, på två olika sätt
- Hårda länkar
  - Här pekar man till en viss i-node direkt. Fungerar bara på samma partition.
  - Oavsett vilken man tar bort så finns den andra kvar
- Mjuka eller symboliska länkar
  - Man ger sökvägen till en fil/katalog
    - Tar man bort originalet så fungerar inte länken längre.
  - Flytta filer
    - Symboliska länkar kan vara bra om man behöver "flytta" en katalog till en annan disk.

ID

- Varje process har ett antal ID:n, som används för att avgöra vad man får göra, i operativsystemet.
- Real user ID, real group ID
  - Talar om vem som "äger" processen och vilken grupp den "tillhör". Dessa kommer från den som startat processen och kommer i sin tur från passwd filen vid inloggning.
- Effective user och group ID
  - Dessa ID används för att kolla rättigheter osv. I normala fall samma som real, men vissa program kan byta ut dessa för att ge programmet särskilda rättigheter.
- Saved set-user-ID och set-group-ID
  - Beskrivs närmare i kapitel 8

FILE ACCESS

- Varje fil har ett antal bitar som talar om vem/vilka som får läsa/skriva i den beroende på vilken användare man är/vilken grupp man tillhör och filens ägare/grupp.
- Olika grupper av rättigheter
  - Man delar upp användarna i tre grupper och man kan sedan ge dem olika rättigheter.
  - De tre grupperna är: user, group och others.
  - När man skriver `ls -l` så ser man hur denna uppdelning är gjord:

```
-rw-r--r-- 1 root other    0 Sep 20 15:30 reconfigured
drwxrwxr-x 2 root sys    512 Aug 25 21:07 sbin/
lrwxrwxrwx 1 root other   11 Jan 10 2000  src -> /import/src/
```

FILE ACCESS

- Den första bokstaven talar om vilken typ av fil det är t ex
  - den första är en vanlig fil '–'
  - den andra är en katalog '`d`'
  - den tredje en symbolisk länk '`l`'
- Sedan följer tre grupper om tre bokstäver, `r` = read, `w` =write och `x` = execute.
  - alla kan läsa filen '`reconfigure`' men bara 'root' kan skriva i den.
  - Alla kan gå in i katalogen '`sbin`' och även läsa den men bara 'root' och de användare som tillhör gruppen 'sys' kan skapa filer i den.
  - Alla kan använda filen '`src`'.

FILE ACCESS – SET-USER-ID

- När en användare startar en process får den en **real UID** som är samma som användarens.
- Normalt är även **effective UID** samma, men om set-user-id på den exekverbara filen är satt så blir EUID samma som filägarens.

FILE ACCESS – SET-USER-ID

- Exempel
- `Fil    Ägare    Rättigheter`
  `filA1    A      -rw-rw-rw-`
  `filA2    A      -rw-------`
  `filB1    B      -rw-------`
- `a.out    A      -r-xr-xr-x`
- `A kommer åt A1 och A2 (ej B1) via a.out`
- `B kommer åt A1 och B1 (ej A2) via a.out`
- `a.out    A      -r-sr-sr-s   obs setUID`
- `B kommer åt A1 och A2 (ej B1 direkt) via`

FILE ACCESS

- `access`
  - Används för att kolla om användaren kan komma åt filen/katalogen.
- `umask`
  - Sätter vilka rättigheter skapade filer ska ha (för den aktuella processen). Accessbitar som är satta här kommer inte sättas vid skapande av filer
- `chmod, fchmod`
  - Funktionernas arg är rättigheterna plus en path respektive en redan öppnad fils fd
  - Unix-kommandot `chmod` kan ta ett oktalt tal med access-bitar, eller mer lättlästa parametrar (tex u owner, g group, o others, a all, r read, w write, x execute …) Exempel:
    `chmod u+r fil` slå på läsrättigheter för owner

FILE ACCESS

- `chown, fchown, lchown`
  - Byter ägare/grupp för en fil
- "hål" i filer uppkommer om man flyttar bortom filslut (med ngn av `seek` funktionerna) och sedan skriver.
  - Hål läses som 0 (eg. en byte som är 0)
- `truncate, ftruncate`
  - Används för att korta av filer.
  - Anger antal bytes man vill ha
    - om större kortas filen
    - om mindre utökas den med '\0'-bytes

KATALOGER

- Läsa kataloger
  - En katalog är egentligen en fil som innehåller information om andra filer och det är alltså möjligt att läsa den för att få fram de andra filerna. Men man gör inte på det vanliga sättet utan man använder speciella funktioner. (Eftersom formatet skiljer mellan olika system.)
  - Garanterat:

```
struct dirent {
  ino_t d_ino;
  char d_name[…];
  ... /* ev fler */
}
```

KATALOGER

- Funktioner för kataloger
  - `DIR *opendir(char *name)`
  - `struct readdir *readdir(DIR *dirp)`
    - Returnerar info om nästa fil i katalogen `dirp`. Obs att värdena skrivs över av nästa anrop till `readdir`, dvs funktionen returnerar samma address.
  - `void rewinddir(DIR *dirp)`
    - Spolar tillbaka `dirp` så att nästa `readdir`-anrop returnerar första filen igen
  - `int closedir(DIR *dirp)`
- Byta katalog
  - Använd funktionerna `chdir` och `fchdir`
  - Om du vill veta var du befinner dig så anropar du `getcwd`

SIGNALER

SIGNALER

- Signaler kan betecknas som mjukvaruavbrott och du kan välja om du vill ta hand om dessa signaler eller strunta i dem. De funkar på ungefär samma sätt som hårdvaruavbrott och du kan definiera rutiner som tar hand om dem.
  - De används oftast för liknande saker som hårdvaruavbrotten, dvs. signalera att något extra har hänt. Exempel på signaler är `SIGALRM` (alarm) och `SIGCHLD` (nåt har hänt med barnet till en process).
- Beroende på operativsystem så finns det flera olika varianter av signaler. Vissa definierar 15 stycken andra 31. För att få reda på den exakta funktionen och vilka som finns blir man tvungen att läsa dokumentationen för det aktuella operativsystemet.

SIGNALER

- Konstanter för varje signal
  - Börjar med `SIG`
  - Heltal > 0

SIGNALER - GENERERING

- Terminalgenererade
  - Trycker en tangentkombination, t ex `ctlr-C (^C) SIGINT`
- Hårdvarugenererade, t ex
  - `SIGFPE` (t ex division med 0, floating point overflow mm)
  - `SIGSEGV` (segmentation violation)
- Mjukvarugenererade, t ex
  - `SIGPIPE` (write till en pipe vars read-ände är stängd)
- Via kill-kommandot
  - Godtycklig signal (t ex `kill -FPE` *`pid`*) `SIGTERM` default
- Via kill-funktionen
  - Godtycklig signal, ev från annan process, måste äga mottagarprocessen