MER IO

TERMINAL-IO

- Mycket man måste hålla reda på med tanke på alla olika varianter av IO enheter man kan tänka sig.
- `stty`
  - Kan vara bra att veta att detta kommando finns, med hjälp av detta så kan du sätta och kontrollera egenskaperna för en terminal.
  - Från ett program kan vi ändra/ undersöka dessa egenskaper mha funktionerna `tcgetattr` och `tcsetattr`
- Info om filer kopplade till terminaler
  - `ctermid` ger sökvägen till den kontrollerande terminalen för processen
  - `isatty` kollar om en `fildescriptor` refererar till en terminal
  - `ttyname` ger sökvägen till en terminal kopplad till en fd

TERMINAL-IO

- Diverse info
  - Termcap är en textfil som beskriver vilka egenskaper som en terminal har. På detta sätt kan man enkelt göra program utan att behöva hårdkoda informationen om olika terminaler ... att sedan hitta definitionen är en annan sak.
  - Terminfo är en vidareutveckling av termcap (egentligen en kompilerad version av termcap)
  - Curses är ett bibliotek som man kan använda sig av för att skriva program som använder sig av "textgrafik"

NON-BLOCKING IO

- Tidigare har vi pratat om "långsam" IO som riskerar att blockera systemet "för evigt".
- `O_NONBLOCK`
  - Genom att definiera detta (vid `open` eller mha `fcntl` för redan öppna filer) så får man funktioner som inte blockerar om det inte finns någon info att läsa eller om det inte är möjligt att skriva.
  - `read` kommer att avslutas med returvärdet `-1` och `errno` satt till `EAGAIN`
  - `write` kommer att returnera den mängd som den lyckades skriva, eller `-1` med `errno` satt till `EAGAIN` om inget data gick att skriva för tillfället

MULTIPLEXING AV IO

- Problemet här är att man vill läsa/skriva till/från ett antal olika filer samtidigt men att data inte alltid finns tillgänglig när man vill ha det. Man vet ej heller vilken fil som är redo först.
  - Det gör att man inte enkelt kan välja vilken fil att använda först.
- **Alt1)** Non-blocking + springa runt och titta
  - En lösning är att sätta alla filer till non-blocking och sedan ”springa runt” och kolla om det finns möjlighet att göra nåt. Nog fungerar det men det är inte precis det mest resurssnåla sättet att göra det på. Kräver också att du försöker gissa tider, storlekar etc., för att få det så bra som möjligt.

MULTIPLEXING AV IO

- **Alt 2)** Asynkron I/O
  - Låt OSet skicka en signal när det finns I/O redo. Stöds ej av all OS och oftast begränsat vilka typer av som tillåts. Dessutom vet vi ej vilken fd det gäller om vi väntar på flera.
- **Alt 3)** Två processer
  - Fungerar också men blir antagligen lite bökigt om man har nåt sammanhang mellan de olika in och utmatningarna (tänk först på problemen att få 2 processer att samverka och skala sedan upp det till 10-20).

MULTIPLEXING AV IO

- **Alt 4)** En bättre möjlighet är att använda någon av multiplex-funktionerna eftersom man då kan bygga listor med de filer vi är intresserade av och sedan överlåta åt OSet att bevaka dessa åt oss.

MULTIPLEXING AV IO – SELECT

1. Skapa en fd mängd
   - Man skapar en mängd med de file descriptors som man är intresserad av.
2. Nolla den
   - För att man inte ska råka bevaka en icke-existerade file descriptor
3. Markera vilka fd man ska kolla
   - Tala om vilka file descriptors man är intresserad av genom att använda `FD_SET` macrot
4. Upprepa med de tre seten
   - Det finns tre olika mängder: läsa, skriva och signaler.

MULTIPLEXING AV IO – SELECT

5. anropa `select`
   - Gör nu själva anropet och låt kärnan ta över bevakningen. Vid anropet så anger man hur länge man vill att kärnan ska vänta på att data blir tillgängligt/kan skrivas

```
int select(int nfds,
           fd_set *readfds,
           fd_set *writefds,
           fd_set *exceptfds,
           struct timeval *timeout);
```

MULTIPLEXING AV IO – PSELECT

```
int pselect(int nfds,
            fd_set *readfds,
            fd_set *writefds,
            fd_set *exceptfds,
            struct timespec *timeout,
            const sigset_t *sigmask);
```

- Skillnader
  1. `timespec` istället för `timeval` (nanosekunder istf mikrosekunder)
  2. Nytt argument `sigmask`
  3. `select` kan uppdatera `timeout`, `pselect` gör det aldrig

MULTIPLEXING AV IO – SELECT

6. vänta på returvärde
   - Nu är det bara att luta sig tillbaka och vänta på att det ska hända nåt
   - `-1` error (kanske en signal)
     - Nåt fel har uppstått. En möjlig orsak är att det har kommit en signal innan någon file descriptor är färdig för att läsa/skriva.
   - `0` time out
     - Tidsgränsen nåddes utan att någon av filerna var klar för läsning/skrivning
   - `>0` hur många fd som är redo
     - Här får man nu reda på hur många file descriptors som är reda att läsas/skrivas. Använd macrot `FD_ISSET` för att undersöka vilken/vilka som är färdiga

MULTIPLEXING AV IO – POLL

- Fungerar principiellt som `select`, men har en annan syntax
-
```
int poll(struct pollfd *fds,
         nfds_t nfds,
         int timeout);
```
-
```
int ppoll(struct pollfd *fds,
          nfds_t nfds,
          const struct timespec *tmo_p,
          const sigset_t *sigmask);
```

```
struct pollfd {
  int   fd;         /* file descriptor */
  short events;     /* requested events */
  short revents;    /* returned events */
};
```

MINNESMAPPADE FILER

- Man kan mappa en fil mot en viss minnesadress. Access till detta minne kommer sedan att läsa/skriva till filen (beroende på hur filen är öppnad mm)
- Åstadkoms mha

```
caddr_t mmap(caddr_t addr, size_t len,
             int prot, int flag,
             int filedes, off_t off)
```

- Mappningen kan tas bort mha

```
int munmap(caddr_t, int len)
```