Om signaler

Synkronisering via signaler
Processer i *Linux/UNIX* kan kommunicera genom att skicka *signaler* till varandra för att meddela att olika händelser inträffat. Det finns olika signaler med namn och nummer. (Se `sigaction(2)` som ger en översikt på över vad signaler är och hur de används i UNIX.) När en process tar emot en signal kan det liknas vid ett avbrott som orsakas av mjukvaran, ett mjukvaruavbrott. Då en process tar emot en signal kommer processen att avbrytas och operativsystemet ser till att en speciell avbrottsrutin körs. Man kan till viss del programmera dessa avbrottsrutiner och man kallar dem i så fall *signalhanterare*. Efter signalhanteraren körts sker ett återhopp till den position som processen hade då signalen mottogs. Kommandot `> kill -l` ger en lista på tillgängliga signaler.

```
Johnny@Concordia:~/osVT2008/lab4> kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
Johnny@Concordia:~/osVT2008/lab4>
```

I fallet med start av barnprocesser med hjälp av `fork()` finns speciella signaler som sänds från barnprocessen till föräldraprocessen. Bland annat `SIGCHLD` som sänds då ett barn avslutas. Det är detta som ligger bakom implementationen av systemanropen `wait()` och `wait4()` mm. Det finns också signaler för pipe som vi ser ovan. (Nr 13. Skickas till en process som försökt skriva på en pipe där ingen läser.) Genom "`info kill`" får ni hela listan.

En process kan ignorera en signal och det är ofta det som sker, en signal skickas till en process och om det finns en signalhanterare så körs den. Dock kan inte signalen 9 - `SIGKILL` ignoreras. Det betyder att processen i fråga dödas.

Förberedelse: Läs igenom avsnittet *3.3 Signals* i *Advanced Linux Programming*, skriv in och kör programmet i listning 3.5. Pröva att se om du kan skicka `SIGUSR1` till processen med kommandot `kill(2)`. Studera manualsidorna till `sigaction()` och `kill(2)`. Läs manualsidan till `dd` och testa exemplet därifrån med att skicka signalen `USR1` till `dd`.

Vi tittar tillbaka på övningen på *sockets*. I början av det programmet stod det så här:

```
void sigchld_handler(int s) {
    while(waitpid(-1, NULL, WNOHANG) > 0);
}
```

respektive, i programmet stod det

```
sa.sa_handler = sigchld_handler; // reap all dead processes
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART;
if (sigaction(SIGCHLD, &sa, NULL) == -1) { perror("sigaction"); exit(1);}
```

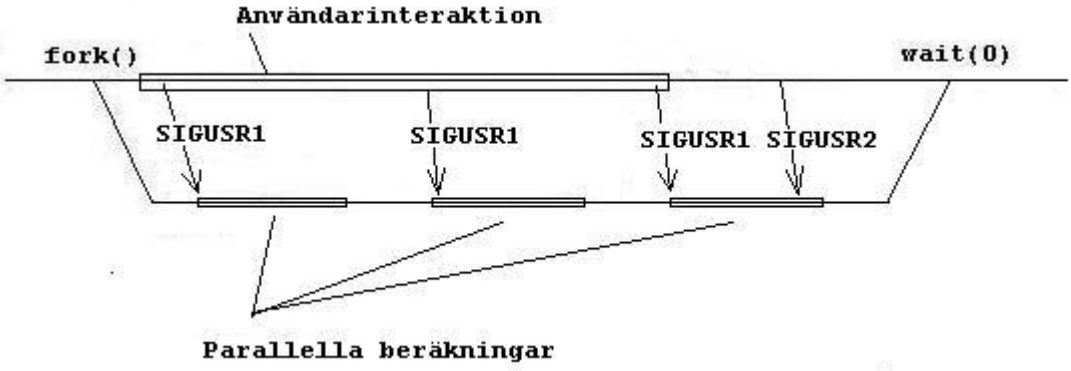
Vad som skedde här är att systemanropet `sigaction()` användes för att installera signalhanteraren `void sigchld_handler(int s)` som körs varje gång signalen `SIGCHLD` skickas till processen. "Processen" i det här fallet är våran server och varje gång ett barn till servern avslutats (det vill säga en klientkommunikation är fullbordad) så körs alltså signalhanteraren. Den kod som finns däri innebär att det barn som avslutats (och är en zombie) inväntas med `waitpid()` som är en variant på `wait()`. Det här är ett alternativt sätt att programmera så att man inväntar barnprocesser så att de inte blir zombier. Det här liknar ju händelsehantering eller hur? Det *är* händelsehantering.

Ett annat sätt att använda signaler tillsammans med `fork()` skulle kunna vara så här:

1. En process startar en interaktion med en användare som innebär en inmatning av data från användaren. Beräkningar ska göras på dessa data som tar väldigt lång tid och samtidigt behövs mer data från användaren.

2. För att utföra dessa beräkningar parallellt med interaktionen med användaren startar processen en barnprocess som väntar på data. Föräldraprocessen kommuniverar med barnet genom att sända signalen `SIGUSR1` för att signalera till barnet att data finns tillgängligt.

3. Då föräldern inte längre behöver den parallella beräkningskapaciteten kan förlädern signalera till barnet att avsluta sig själv med `SIGUSR2`. Då avslutar barnet sig själv och föräldern inväntar barnet i god ordning.



Övning på signaler: Någon av signalerna sänds till en process då användaren trycker *control-C* för att avbryta en process. Ta reda på vilken genom att skriva ett program som fångar upp en signal och experiementera dig fram tills du hittar vilken signal det är. Modifiera sedan programmet så att det inte avslutas direkt utan skriver ut en text: "jaha, jag ska avslutas. Vänta jag ska bara ta ett bad först." sedan sover processen med `sleep()` i 10 sekunder och sedan skrivs det ut "Mmmm, nu slutar jag!" och så avlutas processen. Detta kan symbolisera att man bygger in en säkerhetsspärr då man trycker *control-C*. En klassisk variant är när man trycker på stängningsrutan i ett *Windows*program, då får man frågan "Ska du verkligen inte spara" om man inte sparat den fil man arbetar med. Denna övning representerar ett sätt att göra detta för terminalprogram. (Denna övning redovisas inte men kan vara bra att göra inför tentamen.)