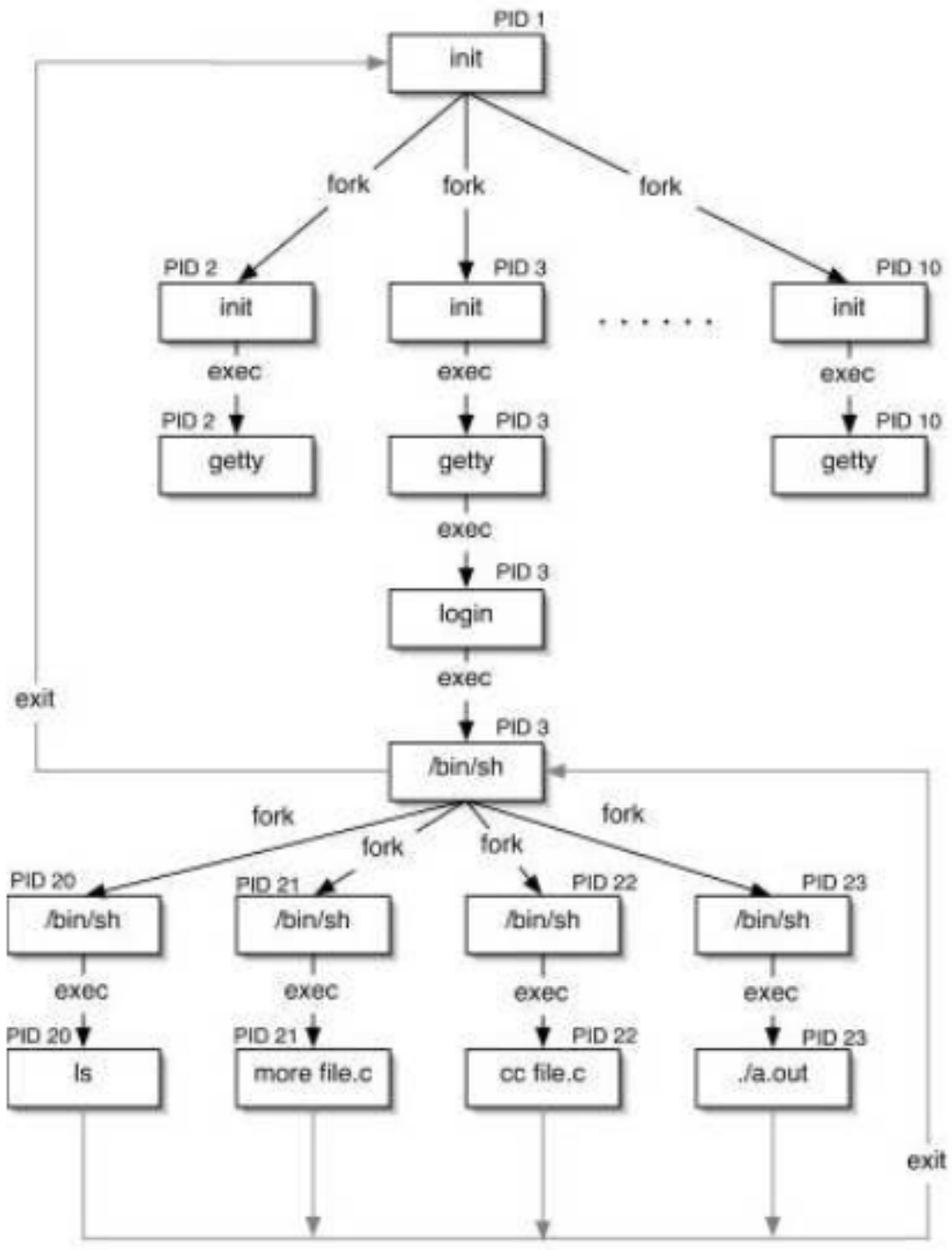
Processi - II

Franco Maria Nardini

Processi

- Programmi in esecuzione in memoria sono chiamati *processi*.
  - Caricati in memoria da una delle sette funzioni **`exec(3)`**.
  - Ogni processo ha un *identificatore univoco* (PID) non negativo.
  - Nuovi processi vengono creati con la system call **`fork(2)`**.
- Il controllo dei processi è fatto principalmente con: **`fork(2)`**, **`exec(3)`**, e **`waitpid(2)`**.

Boot


PID 1
init
fork
fork
fork
PID 2
init
PID 3
init
PID 10
init
exec
exec
exec
PID 2
getty
PID 3
getty
PID 10
getty
exec
PID 3
login
exec
PID 3
/bin/sh
exit
fork
fork
fork
fork
PID 20
/bin/sh
PID 21
/bin/sh
PID 22
/bin/sh
PID 23
/bin/sh
exec
exec
exec
exec
PID 20
ls
PID 21
more file.c
PID 22
cc file.c
PID 23
./a.out
exit

Allocazione di Memoria

- ISO C specifica tre funzioni per l'allocazione di memoria
  - **`malloc`**: alloca uno specifico numero di byte
  - **`calloc`**: alloca la memoria per N oggetti di M bytes
  - **`realloc`**: incrementa o diminuisce la memoria precedentemente allocata:
    - se aumentata: l'area di memoria può essere spostata
    - copie del contenuto (problemi con puntatori!)
    - ritorno della size totale

Allocazione di Memoria

```
#include <stdlib.h>

void  *malloc(size_t size);

void  *calloc(size_t nobj, size_t size);

void  *realloc(void  *ptr, size_t newsize);
```

All three return: non-null pointer if OK, `NULL` on error

Allocazione di Memoria

- implementate con **`sbrk(2)`**:
  - espande o contrae la memoria di un processo
  - **`malloc`** e **`free`** non diminuiscono la memoria:
    - non ritornata al kernel e tenuta nella **`malloc pool`** per allocazioni successive

Segnali

- Sono il mezzo con cui i processi sono notificati di eventi asincroni:
  - un timer settato inizia a suonare (SIGALRM)
  - I/O richiesto è pronto (SIGIO)
  - utente che ridimensiona una finestra (SIGWINCH)
  - utente disconnesso dal sistema (SIGHUP)

Segnali

- Altri modi di generare un evento:
  - segnali generati dal terminale (utente preme una combinazione di tasti che genera segnale)
    - `kill(1)` e (`kill(2)`) abilitano l'utente all'invio di segnali a processi (se owner o superuser)
  - eccezioni hardware (div by 0, reference invalida, ecc).
  - condizioni da software: dati da network file description pronti, ecc.

Segnali

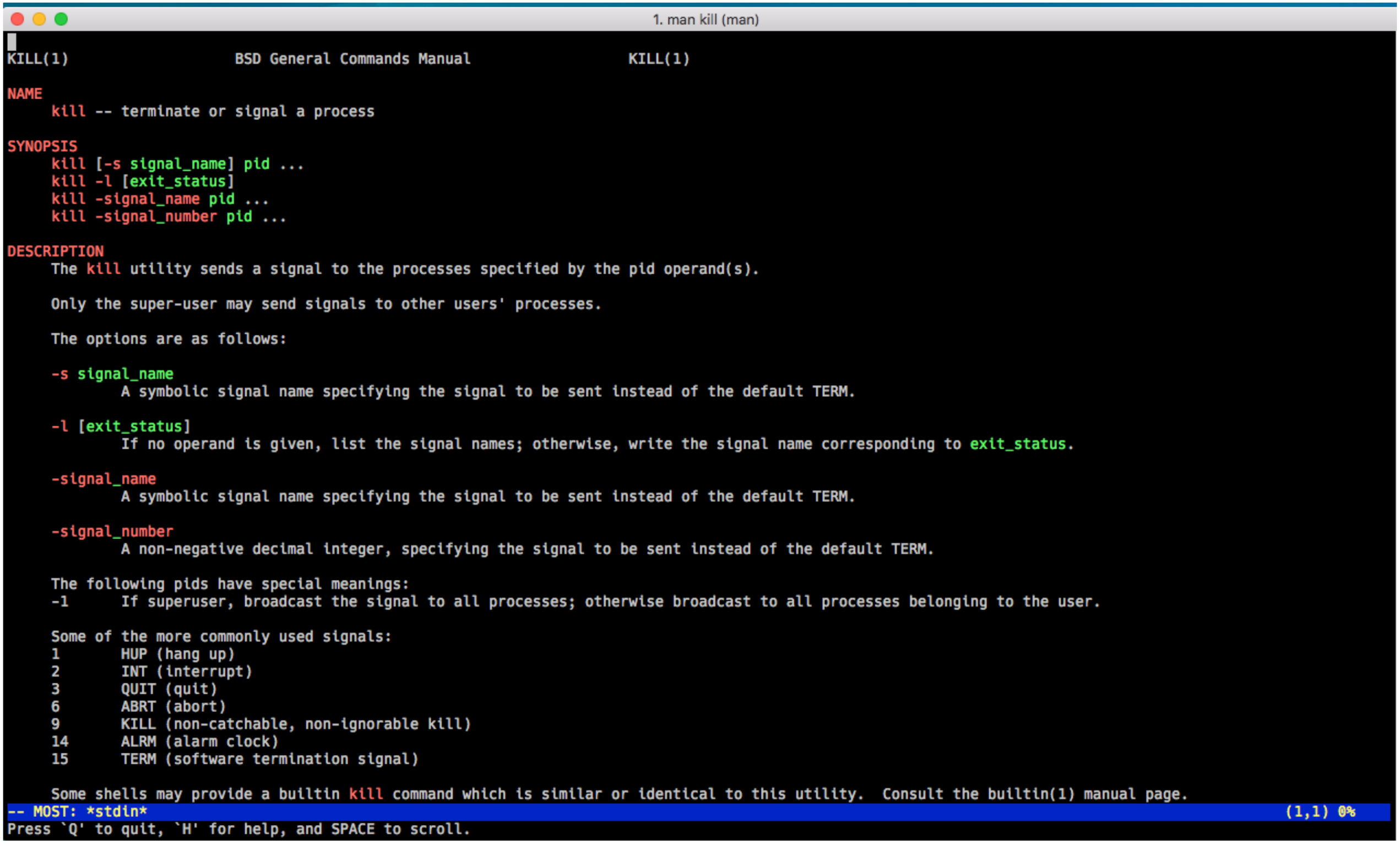
```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int signo);
int raise(int signo);
```

- **`pid`** > 0, il segnale è mandato al processo identificato dal PID
- **`pid`** == 0, il segnale è mandato a tutti i processi aventi PGID uguale a quello del mandante
- **`pid`** == -1
  - POSIX.1 non definisce, BSD si (**`kill(2)`**)

Segnali

- Dopo il ricevimento del segnale, si può fare una di queste cose:
  - Ignorarlo. (ci sono segnali che non si **può** o non si **deve** ignorare)
  - Catturarlo, attraverso una funzione definita da noi che il kernel chiama e che tratta lo specifico caso.
  - Accettare l'azione di default. Il kernel chiama la funzione che implementa l'azione di default per il segnale specifico


1. man kill (man)
KILL(1) BSD General Commands Manual KILL(1)
NAME
kill -- terminate or signal a process
SYNOPSIS
kill [-s signal_name] pid ...
kill -l [exit_status]
kill -signal_name pid ...
kill -signal_number pid ...
DESCRIPTION
The kill utility sends a signal to the processes specified by the pid operand(s).
Only the super-user may send signals to other users' processes.
The options are as follows:
-s signal_name
A symbolic signal name specifying the signal to be sent instead of the default TERM.
-l [exit_status]
If no operand is given, list the signal names; otherwise, write the signal name corresponding to exit_status.
-signal_name
A symbolic signal name specifying the signal to be sent instead of the default TERM.
-signal_number
A non-negative decimal integer, specifying the signal to be sent instead of the default TERM.
The following pids have special meanings:
-1 If superuser, broadcast the signal to all processes; otherwise broadcast to all processes belonging to the user.
Some of the more commonly used signals:
1 HUP (hang up)
2 INT (interrupt)
3 QUIT (quit)
6 ABRT (abort)
9 KILL (non-catchable, non-ignorable kill)
14 ALRM (alarm clock)
15 TERM (software termination signal)
Some shells may provide a builtin kill command which is similar or identical to this utility. Consult the builtin(1) manual page.
-- MOST: *stdin* (1,1) 0%
Press `Q' to quit, `H' for help, and SPACE to scroll.

Esempio

```
$ cc -Wall ../01-intro/simple-shell.c
$ ./a.out
$$ ^C
$ echo $?
130
$ cc -Wall ../01-intro/simple-shell2.c
$ ./a.out
$$ ^C
Caught SIGINT!
```

signal(3)

```
#include <signal.h>

void (*signal(int signo, void (*func)(int)))(int);
```

Returns: previous disposition of signal if OK, SIG_ERR otherwise

- **`func`** può essere:
  - **`SIG_IGN`** che consente di ignorare il segnale **`signo`**
  - **`SIG_DFL`** che consente di accettare l'azione di default per il segnale **`signo`**
  - o l'indirizzo di una funzione che può catturare e gestire il segnale

Esempio

```
$ cc -Wall siguser.c
$ ./a.out
^Z
$ bg
$ ps | grep a.ou[t]
11106 ttys002    0:00.00 ./a.out
$ kill -USR1 11106
received SIGUSR1
$ kill -USR2 11106
received SIGUSR2
$ kill -INT 11106
$
[2]-  Interrupt                  ./a.out
$
```

Startup

- Quando un programma è **`exec`**-guito: :)
  - lo status dei segnali è **`default`** o **`ignore`**
- Quando un programma chiama **`fork(2)`**
  - il figlio eredita la disposizione dei segnali del padre nel caso in cui non ci siano ri-definizioni del comportamento da parte del figlio
- Limite di **`signal(3)`**:
  - si può determinare la disposizione di un segnale solo settandola espressamente

Homework

- pagine man dei contenuti visti

- studiare Stevens, cap. 7, 10