Processes and software packages

Grado en Informática 2025/2026

Departamento de Ciencias de la Computación
y Tecnologías de la Información
Facultad de Informática
Universidad de Coruña

Antonio Yáñez Izquierdo

Contents I

Contents II

Contents III

Managing and monitoring processes

Managing and monitoring processes
→Processes

Processes

- A process is an entity the O.S. uses to execute programs
- A process consists of an address space and one or more threads of control
- Today systems are *multithreaded*, which means that several threads exist inside a process
- In multiprocessor or multicore architectures several threads can run concurrently on different cores

Attributes of processes

- From the system's administrator point of view, the following attributes of processes are to be considered
- **PID** An unique number identifying the process on the system. It is assigned when the process is created.
  - Some systems with container-based virtualization allow for two processes with the same pid to exist concurrently
- **PPID** Identification of the process's parent process
- **Credentials** *real* and *effective uid* and *gid* of the process

Attributes of processes

- **Credentials**
  - The *real* credentials represent the user *'owning'* the process
  - The *effective* credentials define the process privileges
  - Some systems have the *saved* credentials, which are a copy of the *effective* credentials at the start of the process execution
- **control terminal** The terminal associated with the process
  - defines the standard input, standard output and standard error of the process
  - it sometimes affects the delivery of signals
  - daemon processes do not have a control terminal

Attributes of processes

- **priority**. The sheduling priority of a process is a number that defines how much CPU it will get comparatively to other processes
  - Sometimes referred to as *niceness* because it tells how *nice* is the process to other users of the system (high *niceness* $\Rightarrow$ low priority)
  - Priorities are calculated via a dynamic algorithm. Modern systems also have *real time* processes
  - Solaris, linux and FreeBSD have *real time* processes.
    - `priocntl` on Solaris systems
    - `rtprio` on FreeBSD systems
    - `chrt` on linux systems

Attributes of processes

- Open files
- Every process in the system, has *opened* several file system objects: it is using them. At least it has the following
  - directories: current working directory and root directory
  - text file: the program the process is running
  - standard input, output and error
- the commands `lsof`, `pfiles`, `fstat`, `procstat ...` (depending on the system) shows that info

Managing and monitoring processes
→States of processes

States of processes

- A process can be in one of the following states

running	the process is running
runnable	the process can be executed, it will run when scheduled
sleeping	the process is waiting for some resource, it can not be scheduled to run
zombie	the process has finished execution but his status has not yet been collected
stopped	the process is not allowed to execute

Process life cycle

- every process in the system is created by another process, called it's parent process
- the process created is an exact copy of its parent process. It is so until it executes another program (using one of the *exec* system calls)
- the process with pid 1, `init` is the common ancestor of every process on the system (except a few created during system boot)
- some recent linux distros (unfortunately most of them) use *systemd* to manage (among other things) system start up. On those, pid 1 corresponds to *systemd*

Process with pid 1 in diffrent systems

- devuan linux

```
usuario@aso22-1:~$ ps -lp 1
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  80   0 -  3949 -      ?        00:00:01 init
```

- fedora linux

```
[usuario@aso22-2 ~]$ ps -lp 1
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0       1       0  2  80   0 - 27272 -      ?        00:00:01 systemd
```

- solaris

```
 bash-3.2$ ps -lp 1
 F S    UID   PID  PPID   C PRI NI     ADDR     SZ    WCHAN TTY         TIME CMD
 0 S      0     1     0   0  40 20        ?    639        ? ?           0:00 init
```

- freebsd

```
usuario@aso22-1$ ps -lp 1
UID PID PPID C PRI NI  VSZ  RSS MWCHAN STAT TT     TIME COMMAND
  0   1    0 0  24  0 9952 1004 wait   SLs   -  0:00.01 /sbin/init -
```

Process with pid 1 in diffrent systems

- openbsd

```
usuario@aso22-1$ ps -lp 1
  UID   PID  PPID CPU PRI  NI   VSZ   RSS WCHAN   STAT   TT       TIME COMMAND
    0     1     0  16  10   0   408   428 wait    I      ??    0:01.01 /sbin/init
```

- netbsd

```
usuario@aso22-3$ ps -lp 1
UID PID PPID CPU PRI NI   VSZ  RSS WCHAN STAT TTY    TIME COMMAND
  0   1    0   0  85  0 21236 1588 wait  Is   ?   0:00.01 init
```

- ubuntu linux

```
usuario@aso22-2:~$ ps -lp 1
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0       1       0  1  80   0 - 25574 -      ?        00:00:02 systemd
```

Process life cycle

- when a process terminates it supplies an exit code, which can be used to notify why it has terminated. By convention, 0 represents normal termination
- before a process is completely eliminated from the system, the kernel requires that its return code be received by the process's parent (which the parent does with a call to one of the *wait* system calls). The process is kept in a *zombie* state until its parent receives its return code
- if a process terminates before its children, its children are inherited by `init`

Managing and monitoring processes
→Managing processes

Tools to get info on processes

- we can get info on the running processes in one system with the command **ps**
  - the options to ps are not standard. To get (complete) information about ALL the processes `ps -elf` on linux and Solaris and `ps -aux` on BSD systems
- **top** displays information on the running processes on a system on a dynamic way (not a snapshot as **ps** does)
- Solaris systems also have the utility **prstat** to dynamically display information on the running processes
- The comand **pgrep** and **pkill** deal with processes by name of the program being executed, but are not available in every system

Information on processes

- the most common information we get with the **ps** command is

USER	username of the process's owner
PID	Process ID
PPID	Parent process ID
STAT	Process status
%CPU	Percentage of the CPU this process is using
%MEM	Percentage of real memory this process is using
VSZ	Virtual size of the process

Information on processes

RSS	Resident set size (number of pages in memory)
TTY	Control terminal ID
NI	Nice value or SY for system processes
WCHAN	Address of the event the process is waiting for
TIME	CPU time consumed
COMMAND	Command and arguments

ps -aux in an BSD system

```
USER       PID %CPU %MEM    VSZ    RSS TT  STAT  STARTED      TIME COMMAND
_x11     23935  1.0  0.8 11392 16112 ??  Ss     1:42PM    0:02.30 /usr/X11R6/bin/X :0 vt05 -auth /etc/X11/
root         1  0.0  0.0   548   372 ??  Is     1:42PM    0:00.02 /sbin/init
_dhcp    13710  0.0  0.0   620   256 ??  Is     1:42PM    0:00.00 dhclient: em0 (dhclient)
root     26741  0.0  0.0   348   728 ??  Is     1:42PM    0:00.01 syslogd: [priv] (syslogd)
_syslogd 17600  0.0  0.0   356   732 ??  S      1:42PM    0:00.03 /usr/sbin/syslogd -a /var/www/dev/log -a
root     25909  0.0  0.0   484   436 ??  Is     1:42PM    0:00.01 pflogd: [priv] (pflogd)
_pflogd   3762  0.0  0.0   548   328 ??  S      1:42PM    0:00.10 pflogd: [running] -s 160 -i pflog0 -f /v
root      9968  0.0  0.1   640  1148 ??  Is     1:42PM    0:00.01 /usr/sbin/sshd
root      5829  0.0  0.1  1184  1544 ??  Ss     1:42PM    0:00.07 sendmail: accepting connections (sendmai
root     26837  0.0  0.0   292   772 ??  Is     1:42PM    0:00.01 /usr/sbin/inetd
_sndio   14573  0.0  0.0   324   416 ??  I<s    1:42PM    0:00.00 /usr/bin/sndiod
root     28162  0.0  0.0   544   856 ??  Ss     1:42PM    0:00.02 /usr/sbin/cron
root     29701  0.0  0.1   664  1524 ??  Is     1:42PM    0:00.02 /usr/X11R6/bin/xdm
root     23230  0.0  0.1  2060  1108 ??  I      1:42PM    0:00.01 X: [priv] (Xorg)
root     27284  0.0  0.2  1152  4520 ??  Is     1:42PM    0:00.30 xdm: :0 (xdm)
root     30807  0.0  0.0   364   776 ??  I      1:42PM    0:00.00 xconsole
_x11      6018  0.0  0.1   488  2504 ??  I      1:42PM    0:00.04 xconsole
antonio  19706  0.0  0.0   560   476 ??  Is     1:44PM    0:00.03 /bin/sh /etc/X11/xdm/Xsession
antonio  11902  0.0  0.1   868  2396 ??  S      1:44PM    0:00.12 /usr/X11R6/bin/fvwm
antonio   4664  0.0  0.3  3512  5944 ??  R      1:44PM    0:00.17 /usr/X11R6/bin/xterm
antonio   8848  0.0  0.1   524  1616 ??  S      1:44PM    0:00.04 /usr/X11R6/lib/X11/fvwm/FvwmPager 7 4 /u
antonio  11138  0.0  0.0   540   492 p1  Ss     1:44PM    0:00.04 -ksh (ksh)
antonio  19357  0.0  0.0   356   284 p1  R+     1:45PM    0:00.00 ps -aux
root     21615  0.0  0.0   552   376 C0- I      1:42PM    0:00.01 dhclient: em0 [priv] (dhclient)
root     28789  0.0  0.0   472   812 C0  Is+    1:42PM    0:00.02 /usr/libexec/getty std.9600 ttyC0
root     18339  0.0  0.0   420   812 C1  Is+    1:42PM    0:00.01 /usr/libexec/getty std.9600 ttyC1
```

ps -elf in a solaris system

```
F S      UID   PID  PPID   C PRI NI      ADDR     SZ    WCHAN    STIME TTY          TIME CMD
1 T     root     0     0   0   0 SY         ?      0          12:05:20 ?           0:04 sched
1 S     root     5     0   0   0 SD         ?      0        ? 12:05:17 ?           0:02 zpool-rpool
1 S     root     6     0   0   0 SD         ?      0        ? 12:05:22 ?           0:00 kmem_task
0 S     root     1     0   0  40 20         ?    718        ? 12:05:23 ?           0:00 /usr/sbin/init
1 S     root     2     0   0   0 SY         ?      0        ? 12:05:23 ?           0:00 pageout
1 S     root     3     0   0   0 SY         ?      0        ? 12:05:23 ?           0:37 fsflush
1 S     root     7     0   0   0 SY         ?      0        ? 12:05:23 ?           0:00 intrd
1 S     root     8     0   0   0 SD         ?      0        ? 12:05:23 ?           0:00 vmtasks
0 S   netadm    92     1   0  40 20         ?   1043        ? 12:05:55 ?           0:01 /lib/inet/ipmgmtd
0 S     root    11     1   0  40 20         ?   5149        ? 12:05:27 ?           0:13 /lib/svc/bin/svc.s
0 S     root    13     1   0  40 20         ?   4984        ? 12:05:27 ?           0:39 /lib/svc/bin/svc.c
0 S     root   134     1   0  40 20         ?    442        ? 12:06:02 ?           0:00 /usr/lib/utmpd
0 S    dladm    42     1   0  40 20         ?    965        ? 12:05:41 ?           0:00 /usr/sbin/dlmgmtd
0 S     root   638     1   0  40 20         ?    815        ? 12:06:54 ?           0:00 /usr/lib/inet/in.r
0 S   daemon    77     1   0  40 20         ?   3595        ? 12:05:52 ?           0:00 /lib/crypto/kcfd
0 S   netcfg    47     1   0  40 20         ?    962        ? 12:05:43 ?           0:01 /lib/inet/netcfgd
0 S     root   141     1   0  39  0         ?    661        ? 12:06:02 ?           0:00 /usr/lib/zones/zon
0 S     root   105     1   0  40 20         ?   2417        ? 12:05:57 ?           0:01 /lib/inet/in.mpath
0 S     root   112     1   0  40 20         ?    553        ? 12:05:59 ?           0:00 /usr/lib/pfexecd
0 S  antonio  1393     1   0  40 20         ?  32899        ? 12:09:11 ?           0:01 /usr/lib/wnck-appl
0 S     root   647     1   0  40 20         ?   2747        ? 12:06:56 ?           0:00 /usr/sbin/syslogd
0 S     root   252     1   0  40 20         ?   2835        ? 12:06:07 ?           0:04 /usr/lib/devfsadm/
0 S     root   318     1   0  40 20         ?   2348        ? 12:06:17 ?           0:07 /sbin/dhcpagent
0 O  antonio  1567  1462   0  40 20         ?   2372          15:48:29 pts/1       0:00 ps -elf
0 S     root  1427     1   0  40 20         ?   1800        ? 12:09:21 ?           0:01 /usr/lib/hal/hald
0 S  antonio  1457   738   0  40 20         ?   4321        ? 12:09:36 ?           0:00 /usr/lib/rad/rad -
```

ps -elf in a linux system

```
F S UID        PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root         1     0  0  80   0 -  2659 ?      09:49 ?        00:00:00 init [2]
1 S root         2     0  0  80   0 -     0 ?      09:49 ?        00:00:00 [kthreadd]
1 S root         3     2  0  80   0 -     0 ?      09:49 ?        00:00:01 [ksoftirqd/0]
1 S root         6     2  0 -40   - -     0 ?      09:49 ?        00:00:00 [migration/0]
5 S root         7     2  0 -40   - -     0 ?      09:49 ?        00:00:00 [watchdog/0]
1 S root         8     2  0 -40   - -     0 ?      09:49 ?        00:00:00 [migration/1]
1 S root        10     2  0  80   0 -     0 ?      09:49 ?        00:00:01 [ksoftirqd/1]
5 S root        12     2  0 -40   - -     0 ?      09:49 ?        00:00:00 [watchdog/1]
1 S root        13     2  0  60 -20 -     0 ?      09:49 ?        00:00:00 [cpuset]
1 S root        14     2  0  60 -20 -     0 ?      09:49 ?        00:00:00 [khelper]
1 S root        15     2  0  60 -20 -     0 ?      09:49 ?        00:00:00 [netns]
1 S root        16     2  0  80   0 -     0 ?      09:49 ?        00:00:00 [sync_supers]
1 S root        17     2  0  80   0 -     0 ?      09:49 ?        00:00:00 [bdi-default]
1 S root        18     2  0  60 -20 -     0 ?      09:49 ?        00:00:00 [kintegrityd]
1 S root        19     2  0  60 -20 -     0 ?      09:49 ?        00:00:00 [kblockd]
1 S root        20     2  0  60 -20 -     0 ?      09:49 ?        00:00:00 [kacpid]
1 S root        21     2  0  60 -20 -     0 ?      09:49 ?        00:00:00 [kacpi_notify]
1 S root        22     2  0  60 -20 -     0 ?      09:49 ?        00:00:00 [kacpi_hotplug]
1 S root        24     2  0  60 -20 -     0 ?      09:49 ?        00:00:00 [kondemand]
1 S root        25     2  0  80   0 -     0 ?      09:49 ?        00:00:00 [khungtaskd]
1 S root        26     2  0  80   0 -     0 ?      09:49 ?        00:00:00 [kswapd0]
1 S root        27     2  0  85   5 -     0 ?      09:49 ?        00:00:00 [ksmd]
1 S root        28     2  0  99  19 -     0 ?      09:49 ?        00:00:00 [khugepaged]
1 S root        29     2  0  80   0 -     0 ?      09:49 ?        00:00:00 [fsnotify_mark]
1 S root        30     2  0  60 -20 -     0 ?      09:49 ?        00:00:00 [aio]
1 S root        31     2  0  60 -20 -     0 ?      09:49 ?        00:00:00 [crypto]
```

Terminating processes

- most of the times processes terminate by themselves
- sometimes we have to terminate the execution of a process
- we can do this by sending them a signal with the **kill** command
  - we usually send the *software termination signal* requesting to the process that it terminate
  - we can also send the *KILL* signal, that terminates the process unconditionally

Terminating processes

- for a process in the X11 window environment the command **xkill** destroys the X resources of the process, thus terminating it
- the `killall` command available on linux and Solaris also terminates processes
  - the behaviour of the `killall` command differs greatly in solaris and linux systems

Tracing system calls

What is a process doing?

- with the aforementioned utilities we can get useful info on processes
  - process state
  - user behind the execution of a processes
  - resources the process is using (CPU time, priority . . .)
  - command line
  - process parent process
  - controlling terminal

What is a process doing?

- unfortunately, that information gives us no clues on what the process is *actually doing*
- since we have not the source code of a running process we can not know what it is doing
- however, as the process has to ask the system to perform many tasks (*system calls*), we can ask the system to give us information of the *system calls* the process is making

What is a process doing?

- the utility that reports, among other things, what *system calls* a process is making, is different in different operating systems
  - **truss** in Solaris
  - **strace** in linux
  - **ktrace** in BSD systems. *ktrace* produces a binary file, `ktrace.out`, that can be examined with **kdump** (in NetBSD its is also called **ktruss**)

The /proc filesystem

the /proc filesystem

- the commands **top, ps, vmstat, pstat, procmap, procstat...** provide information on the system processes and memory status
- information on the system and the processes can be obtained from the /proc filesystem
- the /proc filesystem is a virtual filesystem, (of type *proc* or *procfs*) mounted on the /proc directory
- it is used by the system to store information about itself and the running processes

the `/proc` filesystem

- the information stored, as its format, varies greatly form system to system
- the kernel creates the contents of `/proc` files on the fly (as they are read), so most of the files appear to be empty when listed with `ls -l`.
  - The info stored of the files becomes available when we *cat* them on the terminal to see what they actually contain

The /proc filesystem
→/proc filesystem in BSD

the /proc filesystem in BSD

- FreeBSD and openBSD don't create the /proc filesystem by default.
- NetBSD creates it by default
- to have /proc on openBSD or FreeBSD the line

  `procfs  /proc  procfs  rw  0 0`

  should be added to the /etc/fstab file
- OpenBSD dropped support for *procfs* on version 5.7. FreeBSD lastest version (14) does not mount it by default
- Example of the /proc filesystem on OpenBSD

```
bash$ ls -l /proc/15099
-r--r--r--  1 antonio  antonio       0 Oct 13 18:56 cmdline
-r-xr-xr-x  3 root     bin      384112 Feb 12  2012 file
-rw-------  1 antonio  antonio  495616 Oct 13 18:56 mem
-r--r--r--  1 antonio  antonio       0 Oct 13 18:56 status
```

the /proc filesystem in BSD

- Example of the /proc filesystem on FreeBSD

```
$ ls -l /proc/971
total 0
-r--r--r--  1 antonio  antonio  0 Nov 11 17:50 cmdline
--w-------  1 antonio  antonio  0 Nov 11 17:50 ctl
-rw-------  1 antonio  antonio  0 Nov 11 17:50 dbregs
-r--r--r--  1 antonio  antonio  0 Nov 11 17:50 etype
lr--r--r--  1 antonio  antonio  0 Nov 11 17:50 file -> /bin/sh
-rw-------  1 antonio  antonio  0 Nov 11 17:50 fpregs
-r--r--r--  1 antonio  antonio  0 Nov 11 17:50 map
-rw-------  1 antonio  antonio  0 Nov 11 17:50 mem
--w-------  1 antonio  antonio  0 Nov 11 17:50 note
--w-------  1 antonio  antonio  0 Nov 11 17:50 notepg
-rw-------  1 antonio  antonio  0 Nov 11 17:50 osrel
-rw-------  1 antonio  antonio  0 Nov 11 17:50 regs
-r--r--r--  1 antonio  antonio  0 Nov 11 17:50 rlimit
-r--r--r--  1 antonio  antonio  0 Nov 11 17:50 status
```

- Example of the `/proc filesystem` on NetBSD

```
usuario@aso22-3$ ls -l /proc/231
-r--------  1 root  wheel    1272 Apr 24 19:31 auxv
-r--r--r--  1 root  wheel       0 Apr 24 19:31 cmdline
lr-xr-xr-x  1 root  wheel       1 Apr 24 19:31 cwd -> /
-r--r--r--  1 root  wheel       6 Apr 24 19:31 emul
-r--r--r--  1 root  wheel       0 Apr 24 19:31 environ
lr-xr-xr-x  1 root  wheel      12 Apr 24 19:31 exe -> /sbin/dhcpcd
dr-x------  2 root  wheel     512 Apr 24 19:31 fd
-r-xr-xr-x  1 root  wheel  322792 Feb 14  2020 file
-rw-------  1 root  wheel     512 Apr 24 19:31 fpregs
-r--r--r--  1 root  wheel       0 Apr 24 19:31 limit
-r--------  1 root  wheel       0 Apr 24 19:31 map
-r--------  1 root  wheel       0 Apr 24 19:31 maps
-rw-------  1 root  wheel  286720 Apr 24 19:31 mem
--w-------  1 root  wheel       0 Apr 24 19:31 note
--w-------  1 root  wheel       0 Apr 24 19:31 notepg
-rw-------  1 root  wheel     208 Apr 24 19:31 regs
lr-xr-xr-x  1 root  wheel       1 Apr 24 19:31 root -> /
-r--r--r--  1 root  wheel       0 Apr 24 19:31 stat
-r--r--r--  1 root  wheel       0 Apr 24 19:31 statm
-r--r--r--  1 root  wheel       0 Apr 24 19:31 status
dr-xr-xr-x  2 root  wheel     512 Apr 24 19:31 task
```

- In NetBSD there are also directories that provide information on the system, such as `cpuinfo`, `meminfo`, `mounts` ...

The /proc filesystem
→/proc filesystem in linux

the /proc filesystem in linux

- contains information on the system and on the processes
- some system parameters can be changed by writing to this files (modern linux systems also support `sysctl` and `/etc/sysctl.conf`)
- apart from the system information directories there is one directory for each process in the system
- we can get info on the processes by examining their directories (in fact this is what the command **ps** does)
- most of the files are text files, that can be *catted* to see the information

a sample /proc filesystem in linux

```
antonio@abyecto:~$ ls /proc/
1     2153  3063  3701  4194  4263  5127  7352       dri           mtrr
10    2158  3066  3705  4200  4271  5181  7494       driver        net
12    2160  3069  3709  4203  4293  581   7495       execdomains   pagetypeinfo
1208  2167  3070  373   4205  4297  6     7496       fb            partitions
13    22    31    3775  4208  4325  603   760        filesystems   sched_debug
14    24    3164  3799  4209  4371  618   7638       fs            self
15    2424  3168  3804  4212  4376  623   7675       interrupts    slabinfo
16    248   3188  3830  4215  4377  681   7681       iomem         softirqs
1661  25    3191  3831  4216  4416  6991  7687       ioports       stat
168   26    3192  3832  4218  4451  7     7696       irq           swaps
169   27    3214  3833  4219  4471  7006  8          kallsyms      sys
17    2781  3307  3834  4220  4474  7028  acpi       kcore         sysrq-trigger
177   2786  3327  3835  4221  4488  7150  asound     keys          sysvipc
178   28    3330  3855  4223  4550  7222  buddyinfo  key-users     timer_list
179   2865  3331  4042  4227  4553  7247  bus        kmsg          timer_stats
18    2889  3338  4065  4229  4555  7249  cgroups    kpagecount    tty
180   29    3368  4116  4232  4594  7250  cmdline    kpageflags    uptime
19    2907  3403  4132  4233  4649  7254  consoles   loadavg       version
1991  3     3404  4175  4239  4954  7259  cpuinfo    locks         vmallocinfo
2     30    3408  4178  4243  4960  727   crypto     meminfo       vmstat
20    3017  3513  4179  4255  4965  7282  devices    misc          zoneinfo
21    3059  3641  4184  4257  4992  7299  diskstats  modules
2122  3061  3669  4192  4262  5126  7306  dma        mounts
```

a sample process directory in /proc filesystem in linux

```
antonio@abyecto:~$ ls /proc/7282
attr          coredump_filter  io          mountstats     personality  statm
autogroup     cpuset           limits      net            root         status
auxv          cwd              loginuid    numa_maps      sched        syscall
cgroup        environ          maps        oom_adj        sessionid    task
clear_refs    exe              mem         oom_score      smaps        wchan
cmdline       fd               mountinfo   oom_score_adj  stack
comm          fdinfo           mounts      pagemap        stat
```

The /proc filesystem
→/proc filesystem in solaris

the /proc filesystem in solaris

- one directory for each process on the system
- the info in this files is mostly in binary format
- Solaris has the utilities in /usr/proc/bin to provide information about the running processes on the system

proc utilities in solaris

pflags Print the /proc tracing flags, the pend- ing and held signals, and other /proc status information for each lwp in each process.

pcred Print or set the credentials (effective, real, saved UIDs and GIDs) of each pro- cess.

pldd List the dynamic libraries linked into each process, including shared objects explicitly attached using dlopen(3C). See also ldd(1).

psig List the signal actions and handlers of each process. See signal.h(3HEAD).

proc utilities in solaris

pstack Print a hex+symbolic stack trace for each lwp in each process.

pfiles Report fstat(2) and fcntl(2) information for all open files in each process. In addition, a path to the file is reported if the information is available from /proc/pid/path. This is not necessarily the same name used to open the file. See proc(4) for more information.

pwdx Print the current working directory of each process.

proc utilities in solaris

pstop Stop each process (PR_REQUESTED stop).

prun Set each process running (inverse of pstop).

pwait Wait for all of the specified processes to terminate.

ptime Time the command, like time(1), but using microstate accounting for reproducible precision. Unlike time(1), children of the command are not timed.

a sample /proc filesystem in solaris

```
bash-3.2$ ls /proc
0     134   245   3     352   366   415   532   618   722   786   817   835   851
1     137   264   321   357   397   416   575   649   727   790   818   841   863
10    184   278   341   360   4     423   583   705   748   792   820   842   944
131   2     283   351   361   404   424   614   720   77    8     834   844
bash-3.2$
```

a sample process directory in the /proc filesystem in solaris

```
bash-3.2$ ls /proc/851
as          ctl         lpsinfo     map         priv        sigact      xmap
auxv        cwd         lstatus     object      psinfo      status
contracts   fd          lusage      pagedata    rmap        usage
cred        ldt         lwp         path        root        watch
bash-3.2$
```

Process privileges and priorities

Process privileges and priorities
→Process privileges

Process privileges

- the process privileges represent what a process in the system can do
  - in relation to files
  - in relation to other processes
- linux implements, to some extent, the draft of POSIX capabilities through *libcap*
- Solaris has its own implementation of a privilege managing system (complete list of process privileges can be got with the command `ppriv -lv`, or `man -s 5 privileges`)

Privileges for accessing the files and the filesystem

- assuming a *'traditional'* UNIX way
  - for the filesystem the *effective credentials* are used
  - some system calls are *privileged*: only a process with effective UID of root can perform them (*mount*, *chown*...)
  - some system calls on one file can only be done by the *user owning* that file (*chmod*)

Privileges for accessing the files and the filesystem

- when a process wants to access a file, the procedure is as follows
  a. **if** the effective user of the process matches the uid of the file, the *user permissions* are used to determine whether the access is granted
  b. **else if** any of the groups of the process matches the gid of the file, the *group permissions* are used to determine whether the access is granted
  c. **else** the *rest of the world* permissions are used to determine whether the access is granted

Privileges for signaling other processes

- traditional UNIX policies state that a signal is delivered
  - if the effective uid of the sending process is that of the *root*
  - if the real or effective uid of the sending process matches the real uid of the receiving process
- SIGCCONT can be delivered to a process in the same session regardless of the uids
- on openBSD a signal is delivered if the real or effective uid of the sending process matches the real uid of the receiving process
- on Solaris and linux a signal is also delivered if the real or effective uid of the sending process matches the real or saved uid of the receiving process

Process privileges and priorities
→Process privileges in Solaris

Process privileges in Solaris

- Solaris provides a more fine mechanism to control what procesess can and cannot do
- A process can have a series of *privileges* which determine which system calls it can perform
- Each process has 4 sets of privileges
  - **effective set** The privileges in effect at a given time
  - **inheritable set** Privileges inherited through an *exec* system call
  - **permitted set** The maximum set of privileges for the process. The *effective set* is a subset of this set
  - **limit set** The upper limit of the set a process and its descendants can have
- the complete set of privileges can be found in `man privileges`

Process privileges in Solaris

```
PRIV_CONTRACT_EVENT
PRIV_CONTRACT_IDENTITY
PRIV_CONTRACT_OBSERVER
PRIV_CPC_CPU
PRIV_DTRACE_KERNEL
PRIV_DTRACE_PROC
PRIV_DTRACE_USER
PRIV_FILE_CHOWN
PRIV_FILE_CHOWN_SELF
PRIV_FILE_DAC_EXECUTE
PRIV_FILE_DAC_READ
PRIV_FILE_DAC_SEARCH
PRIV_FILE_DAC_WRITE
PRIV_FILE_DOWNGRADE_SL
PRIV_FILE_FLAG_SET
PRIV_FILE_LINK_ANY
PRIV_FILE_OWNER
PRIV_FILE_READ
PRIV_FILE_SETID
PRIV_FILE_UPGRADE_SL
PRIV_FILE_WRITE
PRIV_GRAPHICS_ACCESS
PRIV_GRAPHICS_MAP
PRIV_IPC_DAC_READ
PRIV_IPC_DAC_WRITE
PRIV_IPC_OWNER
PRIV_NET_ACCESS
PRIV_NET_BINDMLP
PRIV_NET_ICMPACCESS
PRIV_NET_MAC_AWARE
PRIV_NET_OBSERVABILITY
PRIV_NET_PRIVADDR
PRIV_NET_RAWACCESS
PRIV_PROC_AUDIT
PRIV_PROC_CHROOT
PRIV_PROC_CLOCK_HIGHRES
PRIV_PROC_EXEC
PRIV_PROC_FORK
PRIV_PROC_INFO
PRIV_PROC_LOCK_MEMORY
PRIV_PROC_OWNER
PRIV_PROC_PRIOCNTL
PRIV_PROC_SESSION
PRIV_PROC_SETID
PRIV_PROC_TASKID
PRIV_PROC_ZONE
PRIV_SYS_ACCT
PRIV_SYS_ADMIN
PRIV_SYS_AUDIT
PRIV_SYS_CONFIG
PRIV_SYS_DEVICES
PRIV_SYS_DL_CONFIG
PRIV_SYS_IB_CONFIG
PRIV_SYS_IB_INFO
PRIV_SYS_IP_CONFIG
PRIV_SYS_IPC_CONFIG
PRIV_SYS_LINKDIR
PRIV_SYS_MOUNT
PRIV_SYS_NET_CONFIG
PRIV_SYS_NFS
PRIV_SYS_PPP_CONFIG
PRIV_SYS_RES_BIND
PRIV_SYS_RES_CONFIG
PRIV_SYS_RESOURCE
PRIV_SYS_SHARE
PRIV_SYS_SMB
PRIV_SYS_SUSER_COMPAT
PRIV_SYS_TIME
PRIV_SYS_TRANS_LABEL
PRIV_VIRT_MANAGE
PRIV_WIN_COLORMAP
PRIV_WIN_CONFIG
PRIV_WIN_DAC_READ
PRIV_WIN_DAC_WRITE
PRIV_WIN_DEVICES
PRIV_WIN_DGA
PRIV_WIN_DOWNGRADE_SL
PRIV_WIN_FONTPATH
PRIV_WIN_MAC_READ
PRIV_WIN_MAC_WRITE
PRIV_WIN_SELECTION
PRIV_WIN_UPGRADE_SL
```

Process privileges in Solaris

- Solaris classifies the processes in *Privilege Aware* or *Non Privilege Aware* (traditional processes)
- Privilege Aware processes can manipulate the sets of privileges with the *setppriv* and *setpflags* system calls
- For Non Privilege Aware processes, the effective, inheritable and permitted sets are equal to the *basic* privileges and the limit set is all privileges

Process privileges in Solaris

- Whe can examine the sets of privileges of a process with `ppriv`. `ppriv` can also inform of the privileges missing to perform certain actions
- Privileges can also be assigned to users, roles o right profiles

Process privileges and priorities
→linux process capabilities

linux process capabilities

- linux implements (to some extent) the POSIX 1003-1e capabilities
- these are available as a package and have support in the kernel
- Each process has three sets of capabilities
  - Permitted
  - Effective
  - Inheritable
- and each capability in a set can be enabled or disabled

linux process capabilities

- A capability represents a privilege that can be independtly enabled or disabled (`man capabilities`) lists the capabilities available
- In addition to the functions available in `man libcap` the capabilities package provides the following binaries

getcap	Examines file capabilities
setcap	Sets file capabilities
capsh	A shell wrapper to explore and constrain capability support
getpcaps	Displays the capabilities on the queried process(es)

Process privileges and priorities
→Process priorities

Dynamic priorities

- normal user processes use a dynamic priority system. We'll not deal on the details of the scheduling policies
- although the particular scheduling policies and mechanisms differ from system to system, as far as we are concerned, priorities are calculated dynamically depending, among other factors, on the *niceness* of the process
  - the *niceness* being a number between -20 and 20 with a default value of 0
  - lower values of *niceness* represent greater scheduling priorities

Dynamic priorities

- the command **nice** allows to launch a program with a different *niceness*
- the command **renice** allows to change the *niceness* of an already running process
- only the *root* can decrease the *niceness* of a process

Real-time priorities

- for processes with strict timing requirements, some systems provide real-time priorities: static priorities greater than that of the other processes on the system. Nor the definition neither the implementation are standarized
  - **BSD systems:** FreeBSD implements its own scheme, accessible through *rtprio*, openBSD is said to implement its own soon, based on the POSIX standard. NetBSD implements the POSIX standard accessible only through the system call interface
  - **linux:** has realtime priorities following the POSIX standard accessible through the system call interface: *sched_setscheduler() ...* and the command `chrt`
  - **Solaris:** has several classes of processes depending on how they are scheduled

Real-time priorities in Solaris

- of the several classes that Solaris defines for scheduling, the REAL TIME class is intended for real-time applications
- we can see the classes configured on a Solaris system as well as their characteristics with the command **dispadmin**
- the command **priocntl** allows us to change both the class of one or more processes and their parameters of configuration
- accessing real time classes requires special privileges

Real-time priorities in FreeBSD

- FreeBSD defines different scheduling policies available through the *rtprio* command (or the *rtprio* system call)
- these policies are
  - RTP_PRIO_NORMAL for normal priorities,(dynamically recalculated priorities)
  - RTP_PRIO_IDLE static priorities, smaller than that of normal processes
  - RTP_PRIO_REALTIME real time static priorities, greater than that of normal processes
- accessing real time classes requires special privileges

i/o priorities

- linux implements a *fair-scheduling* algorithm for disk planning
- we can change the input output priority of a process
- this can be accomplished with the command **ionice**. For example, the command

  ```
  bash$ ionice -c 3 -p 5623
  ```

  would lower the i/o priority of process 5623

Signals

Signals
→Signals

signals

- *signals* are methods to notify asynchronous events to processes
- they can be sent among processes as a means of communication
- they can be sent by the terminal driver to kill, interrupt, or suspend processes when keys such as *Control-C* and *Control-Z* are typed
- they can be sent by an administrator or another user (with the `kill` command) to achieve various goals
- they can be sent by the kernel when a process commits an infraction, such as division by zero

signals

- they can be sent by the kernel to notify a process of an *'interesting'* condition such as the death of a child process or the availability of data on an I/O channel
- when a signal is received one of two things can hapen
  - If the receiving process has designated a handler routine for that particular signal, the handler is called. This is often referred to as *'catching'* the signal
  - Otherwise, the kernel takes some default action on behalf of the process. The default action depends on the signal and can be
    - terminate the process (sometimes generating a core dump)
    - do nothing

signals

- A process can also *block* or *ignore* the signal
  - A signal that is ignored is simply discarded and has no effect on the process
  - A blocked signal is queued for delivery at a latter time. The process will not act on it until the signal has been explicitly unblocked
- The handler for a newly unblocked signal is called only once, even if the signal was received several times while reception was blocked

Signals
→Unix common signals

common signals

- INT is sent by the terminal driver when Cntrl-C is typed. It's a request to terminate the process
- TSTP is sent by the terminal driver when Cntrl-Z is typed. It's a request to STOP the process
- STOP stops the process, cannot be caught, blocked or ignored
- KILL terminates the process, cannot be caught, blocked or ignored

common signals

- TERM and QUIT are requests to terminate execution completely. It's expected that the receiving process will clean up its state and exit. QUIT also generates a core dump
- WINCH is used by terminal emulators to indicate a change in their configuration parameters
- SEGV, ILL, FPE indicate execution errors
- USR1 and USR2 are available to programmers

common signals

- HUP usually indicates that the link with the controlling terminal is terminated, causing the process to terminate
  - *csh-like* shells make background processes immune to this signal.
  - in *sh-like* shells this can be done with the **nohup** command
  - traditionally, unix daemons would reread their configuration file upon receiving this signal

common signals

Name	Description	Default	Can catch?	Can block?	Dump core?
HUP	Hangup	Terminate	Yes	Yes	No
INT	Interrupt	Terminate	Yes	Yes	No
QUIT	Quit	Terminate	Yes	Yes	Yes
KILL	Kill	Terminate	No	No	No
BUS	Bus error	Terminate	Yes	Yes	Yes
SEGV	Segmentation fault	Terminate	Yes	Yes	Yes
TERM	Software termination	Terminate	Yes	Yes	No
STOP	Stop	Stop	No	No	No
TSTP	Keyboard stop	Stop	Yes	Yes	No
CONT	Continue after stop	Ignore	Yes	No	No
WINCH	Window changed	Ignore	Yes	Yes	No
USR1	User-defined #1	Terminate	Yes	Yes	No
USR2	User-defined #2	Terminate	Yes	Yes	No

Signals
→Sending signals to processes

Sending signals to processes

- the system administrator can send signals to processes with the command `kill`

  ```
  kill -signal_name  process_pid
  ```

  ```
  kill -signal_number  process_pid
  ```

- the set of available signals varies from system to system and so does the number representing each signal
- we can see the available signals and the associated signal numbers from `bash` with `kill -l`

Signals in openbsd

```
# kill -l
 1    HUP Hangup                      17   STOP Suspended (signal)
 2    INT Interrupt                   18   TSTP Suspended
 3   QUIT Quit                        19   CONT Continued
 4    ILL Illegal instruction         20   CHLD Child exited
 5   TRAP Trace/BPT trap              21   TTIN Stopped (tty input)
 6   ABRT Abort trap                  22   TTOU Stopped (tty output)
 7    EMT EMT trap                    23     IO I/O possible
 8    FPE Floating point exception    24   XCPU Cputime limit exceeded
 9   KILL Killed                      25   XFSZ Filesize limit exceeded
10    BUS Bus error                   26 VTALRM Virtual timer expired
11   SEGV Segmentation fault          27   PROF Profiling timer expired
12    SYS Bad system call             28  WINCH Window size changes
13   PIPE Broken pipe                 29   INFO Information request
14   ALRM Alarm clock                 30   USR1 User defined signal 1
15   TERM Terminated                  31   USR2 User defined signal 2
16    URG Urgent I/O condition        32    THR Thread AST
```

Signals in linux 64 bits

```
antonio@abyecto:~$ 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
```

Signals in Solaris 10

```
bash-3.2$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGEMT       8) SIGFPE
 9) SIGKILL     10) SIGBUS      11) SIGSEGV     12) SIGSYS
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGUSR1
17) SIGUSR2     18) SIGCHLD     19) SIGPWR      20) SIGWINCH
21) SIGURG      22) SIGIO       23) SIGSTOP     24) SIGTSTP
25) SIGCONT     26) SIGTTIN     27) SIGTTOU     28) SIGVTALRM
29) SIGPROF     30) SIGXCPU     31) SIGXFSZ     32) SIGWAITING
33) SIGLWP      34) SIGFREEZE   35) SIGTHAW     36) SIGCANCEL
37) SIGLOST     38) SIGXRES     41) SIGRTMIN    42) SIGRTMIN+1
43) SIGRTMIN+2  44) SIGRTMIN+3  45) SIGRTMAX-3  46) SIGRTMAX-2
47) SIGRTMAX-1  48) SIGRTMAX
```

Signals in Solaris 11

```
bash-3.2$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGEMT       8) SIGFPE       9) SIGKILL     10) SIGBUS
11) SIGSEGV     12) SIGSYS      13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGUSR1     17) SIGUSR2     18) SIGCHLD     19) SIGPWR      20) SIGWINCH
21) SIGURG      22) SIGIO       23) SIGSTOP     24) SIGTSTP     25) SIGCONT
26) SIGTTIN     27) SIGTTOU     28) SIGVTALRM   29) SIGPROF     30) SIGXCPU
31) SIGXFSZ     32) SIGWAITING  33) SIGLWP      34) SIGFREEZE   35) SIGTHAW
36) SIGCANCEL   37) SIGLOST     38) SIGXRES     39) SIGJVM1     40) SIGJVM2
41) SIGRTMIN    42) SIGRTMIN+1  43) SIGRTMIN+2  44) SIGRTMIN+3  45) SIGRTMIN+4
46) SIGRTMIN+5  47) SIGRTMIN+6  48) SIGRTMIN+7  49) SIGRTMIN+8  50) SIGRTMIN+9
51) SIGRTMIN+10 52) SIGRTMIN+11 53) SIGRTMIN+12 54) SIGRTMIN+13 55) SIGRTMIN+14
56) SIGRTMIN+15 57) SIGRTMAX-15 58) SIGRTMAX-14 59) SIGRTMAX-13 60) SIGRTMAX-12
61) SIGRTMAX-11 62) SIGRTMAX-10 63) SIGRTMAX-9  64) SIGRTMAX-8  65) SIGRTMAX-7
66) SIGRTMAX-6  67) SIGRTMAX-5  68) SIGRTMAX-4  69) SIGRTMAX-3  70) SIGRTMAX-2
71) SIGRTMAX-1  72) SIGRTMA
```

Software packages: packages and ports

Software packages: packages and ports
→Software packages

Installing software

- On *windows* systems, when we want to install some software the process is something like this
  - We get hold of the software. Typically an `.EXE` o `.MSI` file
  - Double-click on the file, launching the installer
  - If the file is a compressed file (`.RAR`, `.ZIP` ...) first we extract the files and then launch the installer
  - If the software is on a removable media (for example a CD), we launch the installer from the media (typically named SETUP.EXE, INSTALL.EXE, SETUP.MSI, ...)
- On UNIX systems, although sometimes we may encounter selfstracting scripts or just some software with an installation script, most software is installed via the package system

Package systems

- **What is a package system?**. A set of utilities, together with the appropiate formats that
  - install/uninstall/upgrade/configure software packages
  - keep track of the dependencies an the incompatibilities among different software packages
  - place the executable files, libraries and configuration files at specific locations following system policies
  - perform the necessary actions to integrate the software package in our system
    - including it in the system menus
    - adding it to the list of installed software
    - making administrative tools aware of its presence in our system

Package systems

- the package system also allows for easy removal of a software without leaving unnecessary files behind
- it also helps ensure nothing is deleted by accident, causing software to stop functioning properly
- it provides ready-to-install binaries so no compilation time is needed
- the format and utilities necessary to administer the software packages, that's to say the package system, varies form one unix system to the other

Software packages: packages and ports
→Ports

Ports

- originally appeared in FreeBSD and now common to most BSD systems
- consists of a directory tree with `makefiles` for different software packages
- those makefiles contain instructions on
  - where to fetch the source code
  - what patches to apply
  - how to build the software package from the source
- so, software packages can be built from source with just a couple of commands

Administering software packages and installing software

Tools for administering software packages

- as we seen before, the package system is different for different unix systems
- we'll see the basics of
  - solaris's pkg and IPS systems
  - linux's deb package system
  - linux's rpm package system
  - BSD's pkg package system

Administering software packages and installing software
→Administering software packages in Solaris

Solaris pkg format

- its the traditional way in Solaris systems up to Solaris 10
- The packages reside in directories or in a `.pkg` file
- the basic utilities to manipulate these packages are
  - **pkgadd** adds a package to the system. **pkgadd -d** to specify the location of the package should it not be available at `/var/spool/pkg`
  - **pkginfo** displays software package information, be it the installed packages on the system or a specific package
  - **pkgrm** removes a package from the system

Solaris pkg format

- the package can be a contained in a directory or in a *pkg file*
- if the package is in directory format, the syntax is `pkgadd -d directory name_of_package_directory` (if the directory containing the package directory is `/var/spool/pkg` the -d option can be ommited)
- example

  ```
  # pkgadd -d ./Solaris_i386/Packages/  SFWpdf
  ```

- if the pckage is in a pkg file, we just supply the name of the pkg file to `pkgadd -d`

  ```
  # pkgadd -d ./opera-10.11.gcc4-static-qt3.pkg
  ```

Solaris Image Package System, IPS, format

- Introduced in Opensolaris, is the package system for Solaris 11
- It takes care of both the packages and the system patches in combination with the ZFS boot environments
- Relies on a network centralized repository of packages
- The basic command line utilities are
  - **pkg** Packaging client for general administration of packages
  - *pkgrepo*, *pkgrecv*, *pkgsend*, *pkgdiff*, *pkgmerge*, *pkgmogrify*, *pkgfmt*, *pkgsign*, *pkglint* for package creation and publication

Solaris Image Package System, IPS, format

- there also exists a graphic utility `/usr/bin/packagemanager`
- most of the package administration is done solely with **pkg**

```
NAME
     pkg - image packaging retrieval client

SYNOPSIS
     /usr/bin/pkg [options] command [cmd_options] [operands]

     /usr/bin/pkg install [-nvq] [--accept] [--licenses] [--no-index]
         [--no-refresh] [--deny-new-be | --require-new-be] [--be-name name]
         pkg_fmri_pattern ...
     /usr/bin/pkg uninstall [-nrvq] [--no-index]
         [--deny-new-be | --require-new-be] [--be-name name]
         pkg_fmri_pattern ...

     /usr/bin/pkg update [-fnvq] [--accept] [--be-name name]
         [--deny-new-be | --require-new-be] [--licenses] [--no-index]
         [--no-refresh]

     /usr/bin/pkg refresh [--full] [publisher ...]

     /usr/bin/pkg contents [-Hmr] [-a attribute=pattern ...]
         [-o attribute ...] [-s sort_key] [-t action_type ...]
         [pkg_fmri_pattern ...]
     /usr/bin/pkg info [-lr] [--license] [pkg_fmri_pattern ...]
```

Administering software packages and installing software
→Administering software packages in linux

linux software packages

- linux mainly uses two Software Package Systems
  - **rpm** Introduced by Redhat (**R**edhat **P**ackage **M**anager). It is the standard for Redhat and derivatives: Fedora, Mandrake/Mandriva, Suse ...
  - **deb** It is the standard in *debian* and derivatives.
- On *ubuntu* and *debian* we have the **deb** package system
  - files are in the `.deb` format
  - we have several utilities to deal with deb files (`.deb`): `dpkg`, `apt-get`, `aptitude`, `synaptic` ...

deb package system

- the packages can reside in a central repository or in local media (CD, DVD . . .)
- the location of the packages is described in the file `/etc/apt/sources-list`
- most of the package administration con be done with `apt-get`

`apt-get update`: updates the list of packages available

`apt-get upgrade`: upgrades all the packages to their newest version (if available)

`apt-get install` *package*: installs *package* on the system (together with its dependencies)

`apt-get remove` *package*: removes *package* from the system (and other packages that depend on it)

deb software packages

- there are also other ways to manipulate packages, all of them rely on the contents of `/etc/apt/sources-list` to loacate the packages

`aptitude` analogous to `apt-get` but with a slightly different way of resolving dependencies

`dselect` menu driven utility to deal with packages

`dpkg` utility to deal with the packages individually

`synaptic` debian's graphic front end to the package system. (more graphic front ends are available: *ubuntu's software center* . . .)

fedora rpm packages

- fedora linux (as does redhat, suse and other linux distributions) uses the `rpm` package format
- there's an `rpm` command (similar to `dpkg` in debian linux
- most of the package administration is done through the `yum` utility (similar to `apt-get` in debian)
- from fedora 22 onwards, *dnf* substituted *yum*. The main difference between the two is how they resolve dependencies

fedora rpm packages

- the location of the packages is described in the file `/etc/yum.repos.d` (or where the file `/etc/yum.conf` states)

`yum search`	:	searches the repository
`yum install`	:	installs packages (together with their dependencies)
`yum remove`	:	removes a package (and other packages that depend on it)
`yum update`	:	updates packages
`yum clean`	:	cleans various cache files (used to refresh the list of packages
`yum localinstall`	:	installs a package located locally in the machine

fedora rpm packages

- As of Fedora core 23, the *yum* command has been superseded by the *dnf* utility, which defines an API for extensions and plugins.
- As it maintains an almost complete command line compatibility with *yum*, its basic usage can be summarized

`dnf search` : searches the repository
`dnf install` : installs packages (together with their dependencies)
`dnf remove` : removes a package (and other packages that depend on it)
`dnf update` : updates packages
`dnf clean` : cleans various cache files (used to refresh the list of packages

linux's snap

- *snap* packages include all necessary files to execute a program (libraries, configuration files . . .)
- they are contained in a *squashfs* filesystem
- *snaps* are *sandboxed*
  - they are an easy simple way to distribute packages
  - you end up with a lot o duplicate libraries
  - they are slower to load than standard packages
  - (in principle) they are more secure

linux's snap

- *snap* system consists of

  snapd background service that manages and maintains the snaps on a Linux system

  snap the command line interface used to search, install, remove . . . *snap* in the system

  snap store a repository where to get the *snaps* from and for developers to publish their *snaps*

- *snaps* are distro-independent so, as long as the distro has support for, *snapd* (debian, ubuntu, fedora . . .) *snaps* can be used
- unfortunately *snapd* relies on *systemd* to do some maintenance, thus forcing the user to have systemd installed should they want to use *snaps*

Administering software packages and installing software
→Package administration in BSD systems

openBSD software packages

- BSD systems use the *pkg* format
- the main utilities to perform installation, deinstallation and getting information on packages are

`pkg_add` installs or upgrades software packages
`pkg_delete` removes software packages form the system
`pkg_info` displays information on software packages

openBSD software packages

- the location of packages used to reside in the file `/etc/pkg.conf`, along with other configuration options. More recent versions keep the location to where install from in the file `/etc/installurl`, although this location can be superseded by environment variables
- the location of packages can also be specified with the TRUSTED_PKG_PATH or PKG_PATH environment variables
  - the following lines would install the firefox package from the *rediris* mirror for architecture *i386*

```
# export PKG_PATH=ftp://ftp.rediris.es/mirror/OpenBSD/6.6/packages/i386/
# pkg_add -v firefox
```

FreeBSD software packakes

- the `pkg_add` utility expects to find the packages locally
- if we specify the `-r` option to `pkg_add` the package is to be fetched remotely
- the packages are downloaded from `ftp://ftp.freebsd.org` by default
- to change the default location for the fetching of the packages we can set the environment variables PACKAGEROOT or PACKAGESITE
- A new package management system (*pkgng*) where all the package administration is done through the `pkg` command is being introduced

FreeBSD `pkg` tool

- Appeared in FreeBSD 9.1 and it's the only tool available from FreeBSD 10. onwards, sometimes referred to as *pkg-ng*

`pkg search` : searches the repository
`pkg install` : installs a package (together with their dependencies)
`pkg delete` : removes a package (and other packages that depend on it)
`pkg upgrade` : upgrade from remote repository
`kg autoremove` : removes unwanted dependencies

FreeBSD pkg tool

- pkg tool configuration resides in the files
  - /usr/local/etc/pkg.conf or
  - /etc/pkg/FreeBSD.conf
- Configuration in this files con be overridden by setting one (or more) of the following variables

  MIRROR_TYPE, REPOS_DIR, PACKAGESITE, MIRROR_TYPE, SIGNATURE_TYPE ...

NetBSD `pkg` tool

- As with FreeBSD we have another front end to deal with packages. In NetBSD is calle `pkgin`
- `pkgin` deals with binary packages the same way that `pkg` does in FreeBSD
- it is part of the `pksrc` framework that integrates the packages and the ports
- we can install this utility during initial configuration of the machine or later on using `pkg_add`

Administering software packages and installing software
→The ports system in BSD

The ports system in BSD

- as we saw earlier the ports system provides an alternative way to installing prebuilt packages
- the *ports* is a directory tree structure containing make files for the software packages
- there's one directory for each software package, containing the package descriptions, adecuate makefiles, files checksums,. . .

The ports system in BSD

- this directory structure must be placed in `/usr/ports` (`/usr/pkgsrc` in NetBSD)
  - in OpenBSD this structure is contained in the file `ports.tar.gz` which can be fetched form openbsd's site or any of its mirrors
  - in FreeBSD we install this structure during the installation of the system. If we don't we can get it later with the `'portsnap fetch'` and `'portsnap extract'` commands.
  - in FreeBSD we can also get the `ports.txz` from `/usr/freebsd-dist` in installation disc 1
  - in NetBSD we can install them during initial configuration of the machine or later from `http://ftp.netbsd.org/pub/pkgsrc/stable/pkgsrc.tar.gz`

The ports system in BSD

- once in the directory where the `makefile` is located
  - **make install** installs the software
  - **make fetch** downloads the source files
  - **make package** creates a package that can be installed with **pkg_add**
  - **make** compiles the software
  - **make clean** deletes the files generated during compilation
- the downloaded source files are placed in `'/usr/ports/distfiles'` and the created packages in `'/usr/ports/packages'`

Graphic interface

Graphic interface
→Xorg

Xorg

- Xorg is the most widely extended implementation of the X11 protocol
  - *wayland* defines a protocol of communications between applications and a display server, but as of now, is specific to the linux world
- the X11 protocol has a client/server architecture. The server and the applications need not be in the same machine
- to have a graphic interface in a UNIX type machine we have to have *xorg* installed

Xorg

- xorg can be a part of the base system (for example in OpenBSD) or can be a separate package
- in case it is part of the base system we must install it during installation (or get it from the distribution sets afterwards)
- in case it is part of the package system, we install it with the package system install tool
- once its installed and properly configured we can start the graphical interface with the command *startx*

window managers

- A window manager is an X client than controls the appearance and behaviour of the frames *windows* where graphical applications are drawn
- They determine the window decoration: border, title bar, size ...
- They provide the window buttons, that (usually) enable us to directly terminate the application, resize it, maximize (or minimize) its size ...
- They can also provide menus (sometimes very simple) to start applications and perform operations on the windows
- There are a lot of window managers available: twm, icewm, enlightment, compiz, fvwm, fvwm2, fluxbox, evilwm ...
- We are free to choose which window manager to run

desktop environments

- A desktop environment is a set of software elements that provide a complete user interface in a graphical system
- It consists of icons, toolbars, wallpapers, desktop widgets. a window manager and probably integrated applications and utilities.
- There are many different desktop environments available: Gnome, KDE, CDE, LXDE, xfce, Cinnamon, mate ...
- Some systems (specially some linux distros) provide a Desktop Environment as default, but the system administrator is **free to install** any of them and the user is **free to use** any of the installed

Graphic interface
→Starting the graphic session

startx

- *startx* is a script that starts a graphical section
- it is supposed to be a nicer interface to *xinit*
- it starts the X server and some basic clients
- it runs a script (`/etc/X11/xinit/xinitrc`, `/usr/local/...`) that takes care of the clients to be started
- If the user has a `.xinitrc` it its $HOME directory, then those clients are started

.xinitrc file

- a *.xinitrc* file might look like this

```
#!/bin/sh
program1 [&]
program2 [&]
program2 [&]
....
exec programN
```

- note that if the & is not present, program2 would not start until program1 has finished
- the last program must be called with exec, so that when it ends, the graphical session ends

sample .xinitrc file

- this example of *.xinitrc* file

```
#!/bin/sh
setxkbmap es
xsetroot -solid black
xclock &
xterm &
exec icewm
```

- sets the keyboard type and the background colour, and then starts an xclock, an xterm and the icewm window manager. When the icewm ends, the graphical session ends

sample .xinitrc file

- Here's another example

```
#!/bin/sh
setxkbmap es
xrandr -s 1920x1080
exec mate-session
```

- executes the mate desktop session at fullHD with the spanish keyboard

Graphic interface
→Graphical login

Graphical login

- We can choose, should we want to, to have the machine start in graphic mode and have a graphical login
- Xorg should be already working, as the grapical login is just another program running on X
- we can choose the graphical login program we like. Yet again some linux distros impose a graphical login program, but we can shoose any
- `xdm`, `gdm`, `kdm`, `lightdm`, `slim` ... are alternatives to perform that task

Graphical login

- we can start (or stop) them at anytime with via the script at `/etc/init.d`, `/etc/rc.d`, `/usr/local/etc/rc.d ...` or using `systemdctl` or `svcadm`
- we can also choose to have them started at booting, via the appropiate variables at `rc.conf`, or with the commands `systemctl`, `svcadm`, `update-rc.d`, `insserv ...`
- we can have several of them installed but only one can be running

Graphical login

- Some of the login programs can be configured to allow the user to decide which session he/she wants to start upon login
- Some can be configured to start an specific session
- The user can supply his/her `.xsession` file (same format as `.xinitrc`) to start a spcific session

Virtualization environments

Isolating applications

- the *chroot* system call changes the *root directory* for an application
- it changes the view of the filesystem that the application has
- a *chrooted* application only sees the part of the filesystem it has been chrooted to (name space resolution)

Isolating applications

- unfortunately, should the application have access to the actual devices, with the right privileges it could escape the *chroot* limits
- basically, a virtualized environment consists of a copy of the essential files of the operating system installation to a directory where a chrooted copy of the operating system runs with the devices virtualized

virtualization environments

- we can protect the system from applications by
  - limiting applications resource usage,
  - limiting what part of the filesystem they see through *chroot*
- the next step in isolating the O.S. from possible application *'malfunction'* is having it run in a virtualized environment (VE)

virtualization environments

- an VE is different from a Virtual Machine (as created by tools like VirtualBox or VMWare) in that it requires much less resources and overhead as the VM includes the entire OS and machine setup, including hard drive, virtual processors and network interfaces
- processes running in the VE (usually called container) only see the part of the O.S. file system assigned to it (via *chroot*) and the devices allocated to the container
- we usually refer to this as *container based virtualization*, as the first widespread implementation was the *solaris containers*.

virtualization environments

- compared to VMs, containers generally offer less isolation because they share portions of the host kernel and operating system instance.
- most unix-like O.S.s offer their own brand (or brands) of container based virtualization
- we'll see briefly
  - FreeBSD jails
  - solaris zones (containers)
  - linux LXC containers

Virtualization environments
→FreeBSD jails

Creating a jail

- first we create a directory in which the jail is going to reside. Example

```
# mkdir -p  /usr/jail/JAULILLA
```

- we now extract the base FreeBSD system (and the ports collection, should we want to) in this directory, so, assuming the FreeBSD installation disc-1 is mounted in `/media/12_0_RELEASE_AMD64_CD`, we issue the following commands

```
# cd /usr/jail/JAULILLA
# tar xvJf /media/12_0_RELEASE_AMD64_CD/usr/freebsd-dist/base.tx
# tar xvJf /media/12_0_RELEASE_AMD64_CD/usr/freebsd-dist/ports.t
```

Creating a jail

- For FreBSD versions newer than 14.3, tha distribution sets (`base.txz, base-dbg,txz ...`) cannot be found in the *bootonly disc* nor in the *disc1*
  - they can be found on the *dvd1*
  - we can download them from the repository

`https://download.freebsd.org/releases/amd64/amd64/15.0-RELEASE`

- the rest of the procedure for creating a jail is the same

Creating a jail

- We usually define all the jails in `/etc/jail.conf`
- We can also use (from FreeBSD 14) the directory `/etc/jail.conf.d`, that allows us to define each jail in a separate `.conf` file
- Should that be the case we must include in `/etc/jail.conf`

```
.include "/etc/jail.conf.d/*.conf";
```

- What we describe here is what is called *thick* jails. FreeBSD now also supports other type of jails: thin jails, service jails, VNET jails and linux jails. But the *thick* jails are the ones providing more isolation

Creating a jail

- item, we now define the jail in `/etc/jail.conf`, as in the following example

```
pruebajail {
        path = /usr/jail/JAULILLA;
        mount.devfs;
        host.hostname = jailcilla;
        ip4.addr = 10.0.2.25;
        interface = em0;
        exec.start = "/bin/sh /etc/rc";
        exec.stop = "/bin/sh /etc/rc.shutdown";
}
```

Using a jail

- we can start now the jail with the jail command

```
# jail -c pruebajail
```

- jails can also be started with 'service jail start *jailname*' and stopped with 'service jail stop *jailname*'
- if we want jails to be started at boot time we use jail_enable="YES" in /etc/rc.conf
- jailed processes are shown with J in ps lists
- we can also use the commands 'jls' to list jails and 'jexec' execute commands in jails

Virtualization environments
→Solaris zones

Solaris Zones

- also called containers. Available from Solaris 10
- Solaris distiguishes two types of zones
  - *branded* zones that contain alternative runtime behaviors (Solaris8, Solaris9, linux, cluster zones)
  - *unbranded* zones use the same O.S. that is in the global zone
- The global zone is the default operating system and has control over all the processes. A global zone always exists even when no other zones are configured.
- Non-global zones, or simply zones , are configured inside the global zone. Zones are isolated from the physical hardware. A zone cannot detect the existence of any other zones.

Solaris Zones

- Booting the global zone is equivalent to booting the system hardware.
- Each zone, including the global zone, is assigned a zone name. The global zone always has the name "global".
- Each zone is assigned a unique numeric identifier. The global zone always has the identifier ID 0.

Solaris Zones

- Each zone has a path to its root directory that is relative to the global zone's root directory.
- The global zone is the only zone from which a non-global zone can be configured and installed. (FreeBSD jails can be recursive)
- a non global zone can be administered by a role with the *Zone Management* profile

Creating a zone in Solaris 11

- First we create the file system where the zone is to reside with `'zfs create'`

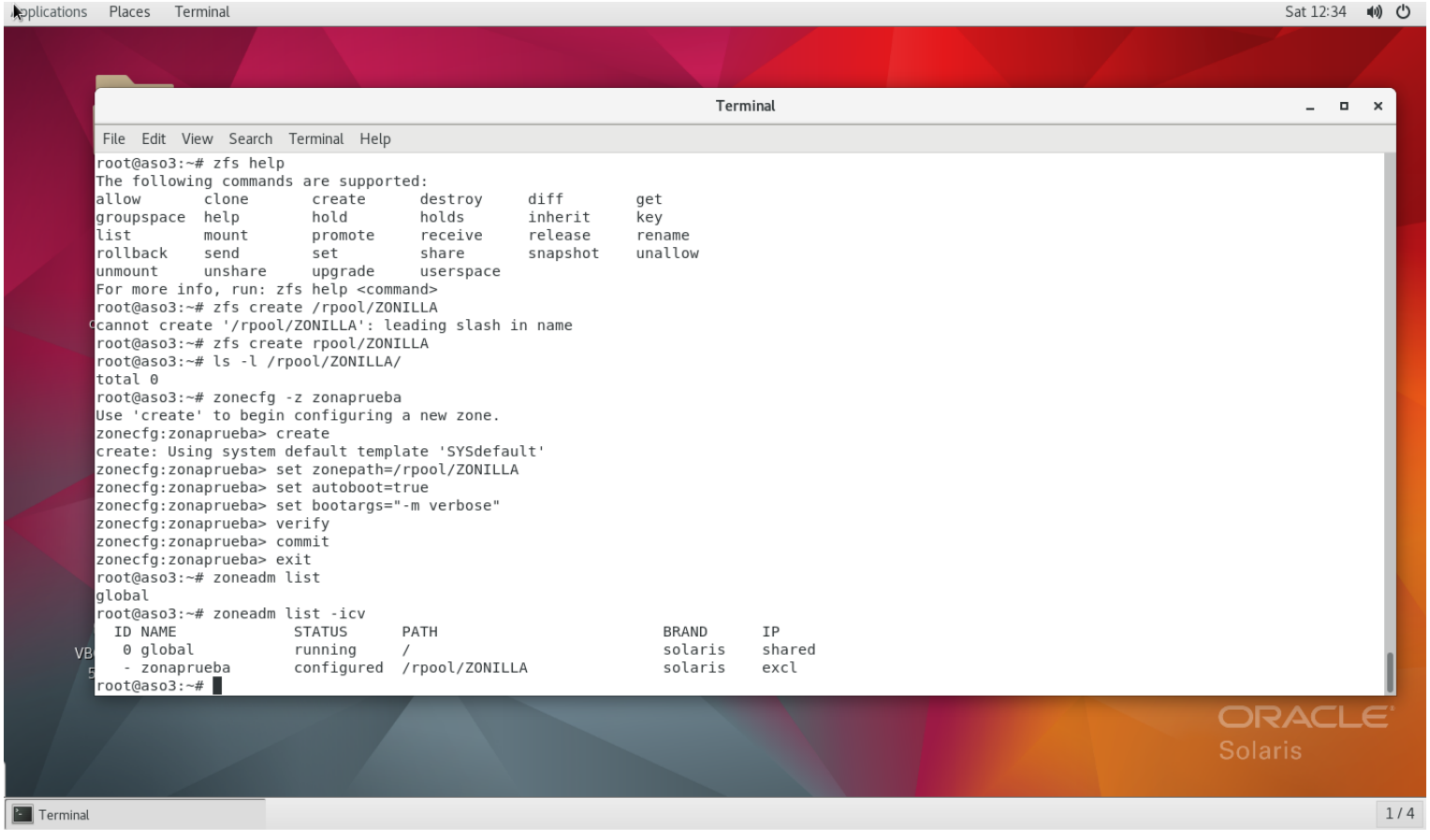
```
root@aso3:~# zfs create rpool/ZONILLA
```

- then we use `'zonecfg'` to create the zone and assign that filesystem to it (we'll name the zone *zonaprueba*)

```
root@aso3:~# zonecfg -z zonaprueba
Use 'create' to begin configuring a new zone.
zonecfg:zonaprueba> create
create: Using system default template 'SYSdefault'
zonecfg:zonaprueba> set zonepath=/rpool/ZONILLA
zonecfg:zonaprueba> set autoboot=true
zonecfg:zonaprueba> set bootargs="-m verbose"
zonecfg:zonaprueba> verify
zonecfg:zonaprueba> commit
zonecfg:zonaprueba> exit
```

- the `'zoneadm list'` command will list the zone

Creating a zone in Solaris 11



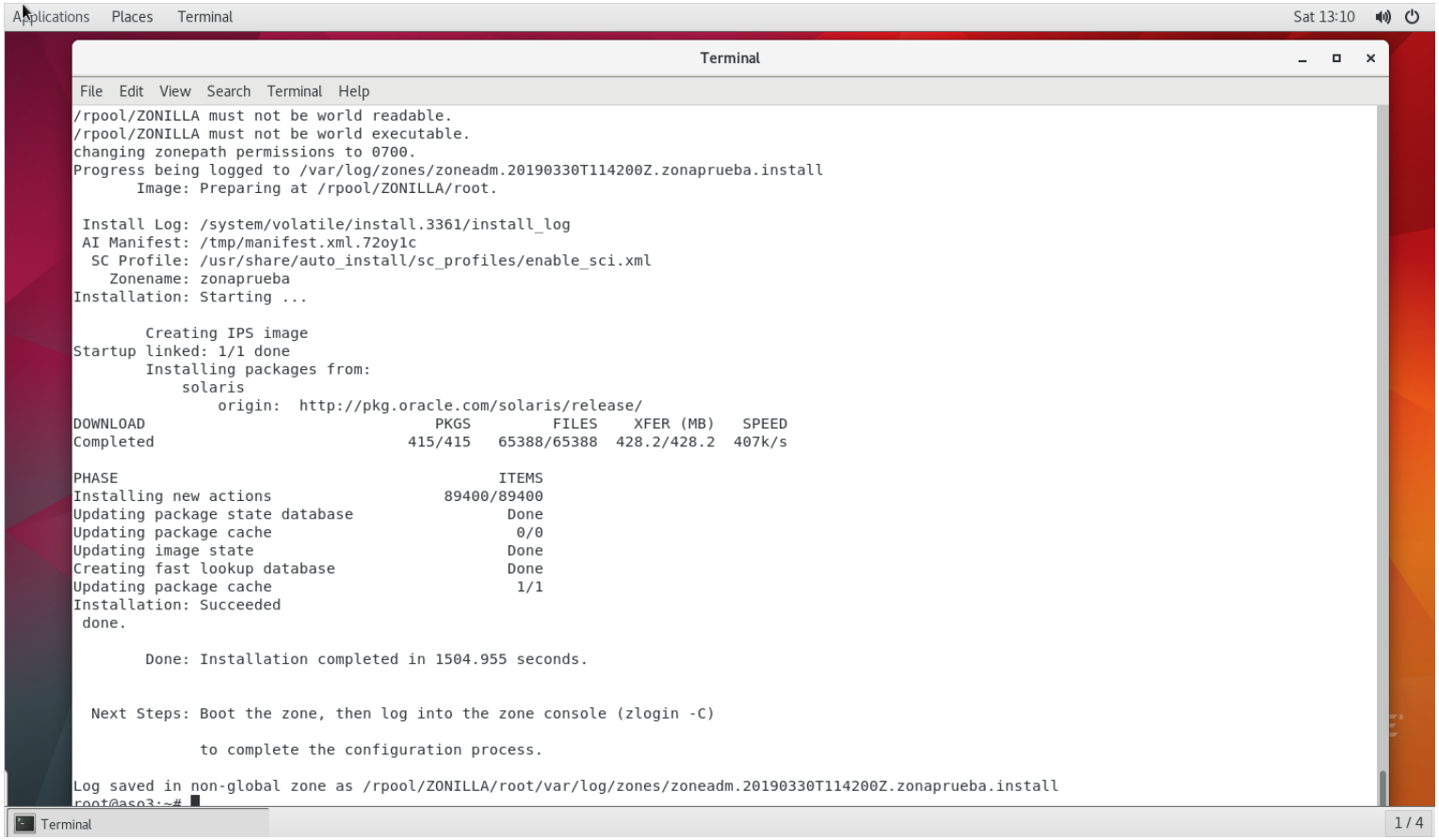
Installing a zone in Solaris 11

- the next step is installing the zone using the command `'zoneadm install'`
- the package repository must be correctly configured as the command `'zoneadm'` will use the package system to install the zone

  ```
  root@aso2:~# zoneadm -z zonaprueba install
  ```

- we can see afterwards with `'zoneadm list -icv'` that the zone is installed

Installing a zone in Solaris 11



Using a zone in Solaris 11

- the next thing is to boot the zone

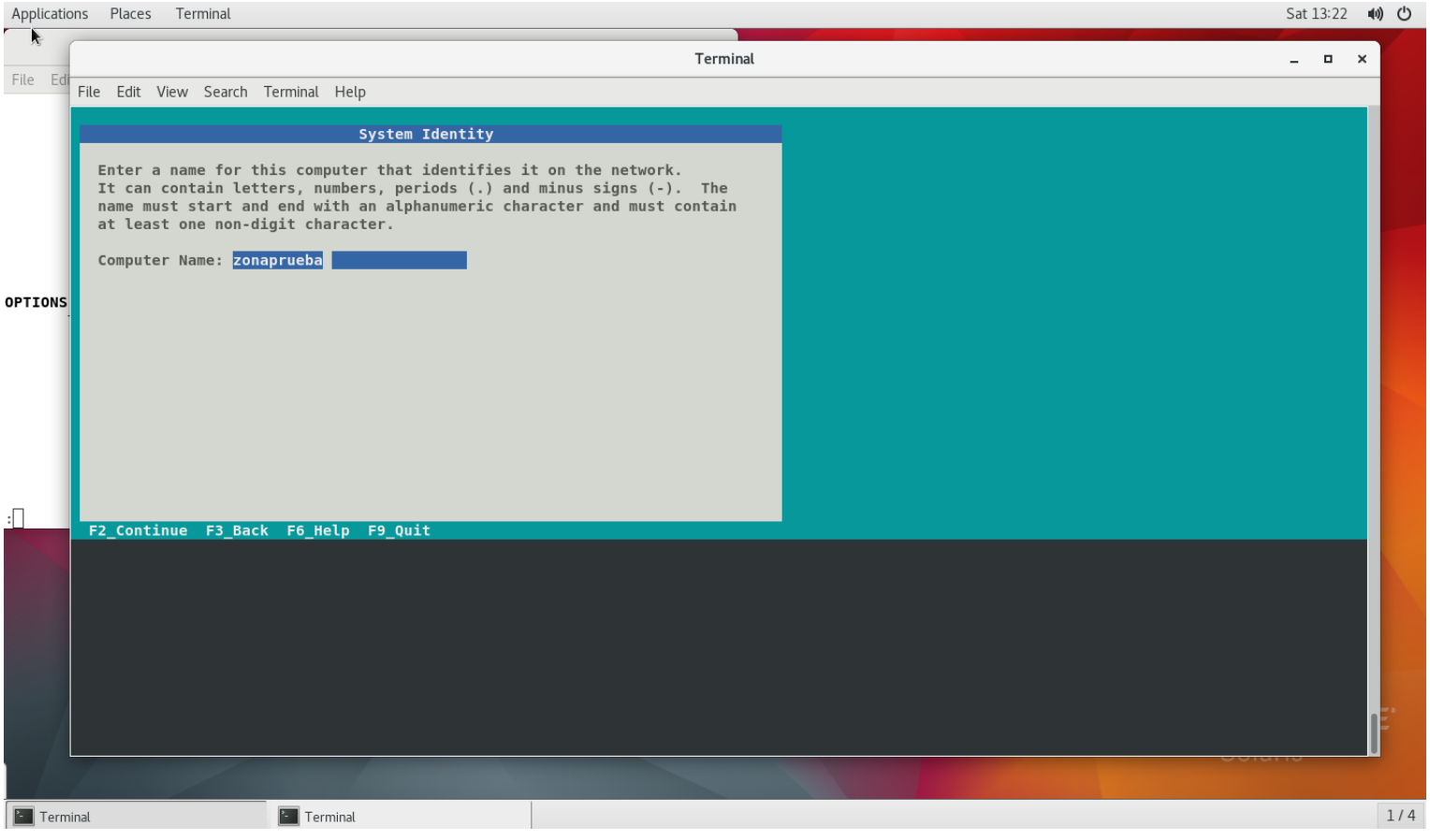
```
root@aso3:~# zoneadm -z zonaprueba boot
root@aso3:~# zoneadm  list -icv
  ID NAME             STATUS      PATH
   0 global           running     /
   1 zonaprueba       running     /rpool/ZONILLA
root@aso3:~#
```

- and to configure the zone by logging into the zone Console

```
root@aso3:~# zlogin -C zonaprueba
```

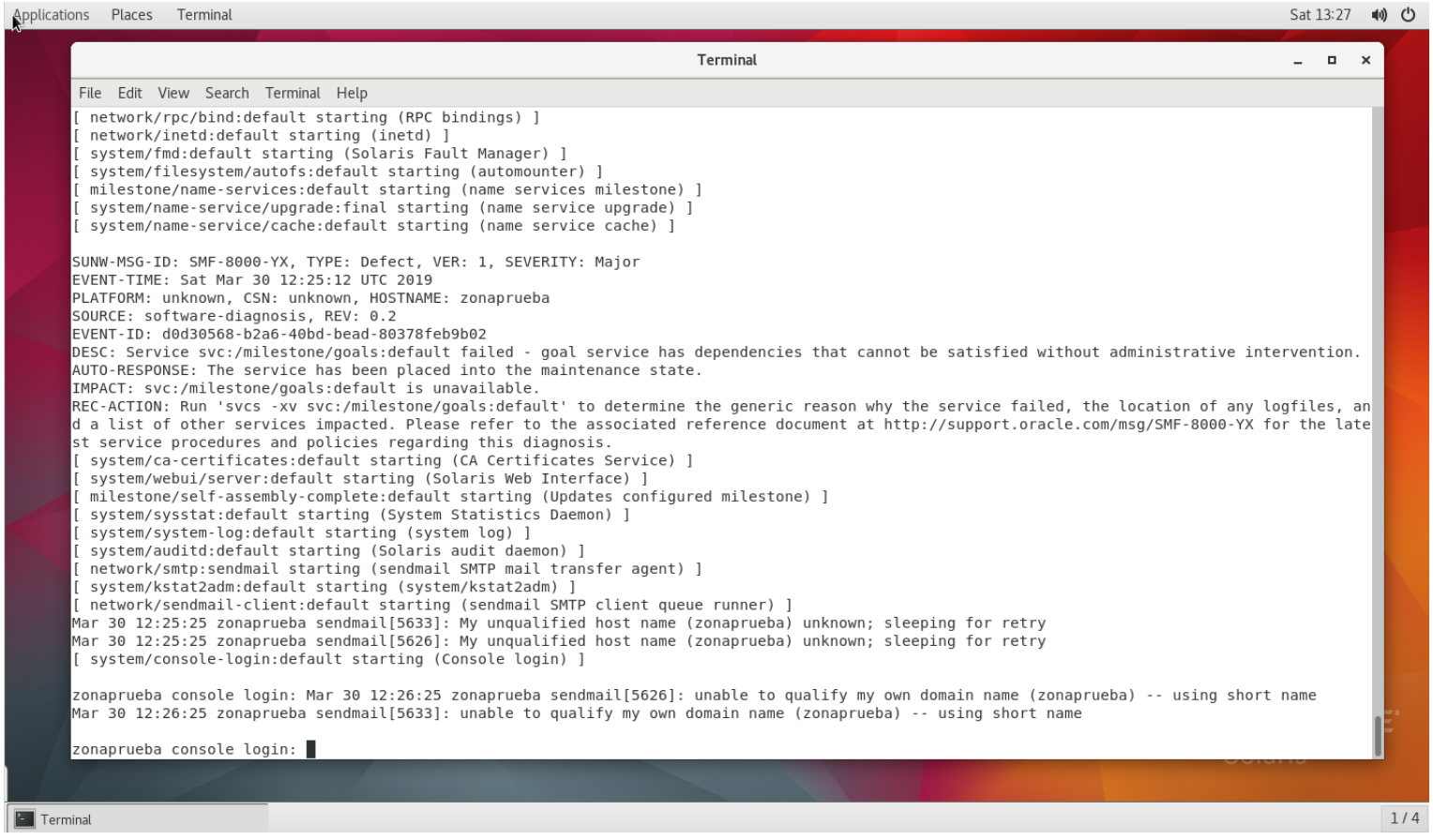
- the first time we login to the zone console, we enter the configuration of the zone (similar to an O.S. installation)

Configuring a zone in Solaris 11



Entering a zone in Solaris 11

Once configured we can login to the zone



Virtualization environments
→linux LXC containers

Creating a container

- we have to install LXC framework and its related packages
- the first thing is to create a container. We just have to provide a name for the container and a template to create the container from
- the name is freely chosen by us and the template is one of the linux flavors in the LXC environment

```
root@abyecto:~# lxc-create -t ubuntu -n PruebaContainers
```

Container templates

- the list of templates available is usually a `/usr/share/lxc/templates`

```
antonio@abyecto:~$ ls -l /usr/share/lxc/templates/
total 408
-rwxr-xr-x 1 root root 13160 Jan 29  2018 lxc-alpine
-rwxr-xr-x 1 root root 13704 Jan 29  2018 lxc-altlinux
-rwxr-xr-x 1 root root 11373 Jan 29  2018 lxc-archlinux
-rwxr-xr-x 1 root root 12159 Jan 29  2018 lxc-busybox
-rwxr-xr-x 1 root root 29725 Jan 29  2018 lxc-centos
-rwxr-xr-x 1 root root 10374 Jan 29  2018 lxc-cirros
-rwxr-xr-x 1 root root 20243 Jan 29  2018 lxc-debian
-rwxr-xr-x 1 root root 17914 Jan 29  2018 lxc-download
-rwxr-xr-x 1 root root 49693 Jan 29  2018 lxc-fedora
-rwxr-xr-x 1 root root 28384 Jan 29  2018 lxc-gentoo
-rwxr-xr-x 1 root root 13868 Jan 29  2018 lxc-openmandriva
-rwxr-xr-x 1 root root 15946 Jan 29  2018 lxc-opensuse
-rwxr-xr-x 1 root root 41791 Jan 29  2018 lxc-oracle
-rwxr-xr-x 1 root root 11570 Jan 29  2018 lxc-plamo
-rwxr-xr-x 1 root root 19242 Jan 29  2018 lxc-slackware
-rwxr-xr-x 1 root root 26862 Jan 29  2018 lxc-sparclinux
-rwxr-xr-x 1 root root  6862 Jan 29  2018 lxc-sshd
-rwxr-xr-x 1 root root 25705 Jan 29  2018 lxc-ubuntu
-rwxr-xr-x 1 root root 11734 Jan 29  2018 lxc-ubuntu-cloud
antonio@abyecto:~$
```

Using the LXC containers

- we start the machine and see that is running ok

```
root@abyecto:~# lxc-ls -f
NAME            STATE   AUTOSTART GROUPS IPV4 IPV6
PruebaContainer STOPPED 0         -      -    -
root@abyecto:~#
root@abyecto:~#
root@abyecto:~# lxc-start -n PruebaContainer
root@abyecto:~# lxc-ls -f
NAME            STATE   AUTOSTART GROUPS IPV4 IPV6
PruebaContainer RUNNING 0         -      -    -
root@abyecto:~#
```

Using the LXC containers

- we start the machine in the foreground with -F
- to manipulate the machine we can use the lxc-* commands

```
 root@abyecto:~# lxc
lxc-attach       lxc-checkpoint   lxc-create       lxc-freeze       lxc-monitor      lxc-unfreeze
lxc-autostart    lxc-config       lxc-destroy      lxcfs            lxc-snapshot     lxc-unshare
lxc-cgroup       lxc-console      lxc-device       lxc-info         lxc-start        lxc-usernsexec
lxc-checkconfig  lxc-copy         lxc-execute      lxc-ls           lxc-stop         lxc-wait
root@abyecto:~# lxc
```

Using LXC containers

- if you want to run lxc as a normal user you have to
  1. add the following lines to file `.config/lxc/default.conf`

     ```
     lxc.id_map = u 0 100000 65536
     lxc.id_map = g 0 100000 65536
     ```
  2. add the line `kernel.unprivileged_users_clone=1` to the file `/etc/sysctl.d/local.conf` and then execute `sysctl --system`
  3. change the permissions of `.local` and `.local/share` to *rwxr-xr-x*
  4. use the download template

Using the LXC containers

- there are other container based virtualization solutions for linux
- the two most widespread are
  - LXD
  - docker
- both of them rely on *cgroups* and *lxc* libraries