Universidad Tecnológica Nacional – Facultad Regional Buenos Aires
Ingeniería en Sistemas de Información
Sistemas Operativos (08-2027)

Sistemas Operativos

Trabajo Práctico Nro.1 – 2do Cuatrimestre 2010

Introducción a Linux y Solaris

Revisión 0.5

AGRADECIMIENTO: a Pedro Vázquez por colaborar en la sección de Solaris aportando un material excelente.

¡Muchas Gracias!

Trabajo Práctico Nro. 1

Índice

1. Introducción 2
2. Objetivos 2
3. Características 2
4. Primera Parte: Introducción a Unix 2
   - 4.1. Ayuda 2
   - 4.2. Teclado / Terminales 4
   - 4.3. Sistema 4
   - 4.4. Usuarios 4
   - 4.5. Archivos 5
   - 4.6. Permisos 5
   - 4.7. Directorios 6
   - 4.8. Filtros 7
   - 4.9. Redireccionamiento 7
   - 4.10. Pipelines 7
   - 4.11. Vim 8
   - 4.12. Montar Dispositivos / Filesystems 8
5. Segunda Parte: Solaris 1
   - 5.1. Información del sistema 1
   - 5.2. Información sobre los procesos en el sistema 4
   - 5.3. Estadísticas del Kernel 13
   - 5.4. Entrada – Salida 13

1. Introducción

Este trabajo práctico pretende introducir al alumno al uso, configuración y breve administración del sistema operativo que será usado como plataforma para los trabajos prácticos posteriores.

2. Objetivos

Adquirir los conocimientos básicos necesarios para poder usar un sistema operativo moderno de tipo UNIX, como lo es Linux.
Conocer los comando más importantes sobre administración de Solaris.
Conocer y comprender las distintas herramientas de administración y configuración de dicho sistema operativo.
Generar una base de información necesaria para la elaboración de los trabajos prácticos posteriores.

3. Características

Duración estimada para su desarrollo: 1 semana
Fecha de finalización: Presentación del TP2
Modalidad de entrega: **OBLIGATORIO**
Modo de entrega: envío por email a todos los coordinadores generales:
http://www.tpsosutnfrba.com.ar/?page_id=54
Dudas y consultas del TP1: http://www.tpsosutnfrba.com.ar/foro/t-tp1-2c2010--11
Modalidad de desarrollo: individual

4. Primera Parte: Introducción a Unix

4.1. Ayuda

4.1.1. man: es un programa que formatea y muestra la páginas del manual de referencia del sistema. El formato de uso básico es "man tema" donde tema es el nombre de la página del manual que se quiere ver.

4.1.1.1. ¿Qué tipo de información provee el man, como la organiza internamente y como busca dentro de la misma? Para saberlo, tipee "man man" (sin comillas), use los cursores UP y DOWN para recorrer la pagina.

4.1.1.2. Salga de la página anterior tipeando "q".

4.1.1.3. Investigue que hace el comando ls tipeando "man ls".

4.1.1.4. Liste de manera ordenada (por tamaño del archivo) mediante el comando "ls" la mayor cantidad de información posible sobre todos los archivos que se encuentren en su directorio home (ej: /home/guest) incluyendo aquellos que empiezan con un punto.

4.1.1.4.1. Tip: cuando se está viendo una página del manual, puede buscar cadenas tipeando "/cadenaABuscar" sin comillas.

4.1.1.4.2. Tip: para posicionarse dentro de un directorio debe hacer "cd directorio". Ejemplo: cd /usr/local

4.1.1.5. Supongamos que se desea conocer el prototipo de la función de ANSI C printf(). Tipee "man printf" y vea que sucede. ¿Es la página que estábamos buscando?

4.1.2. whatis

4.1.2.1. Investigue que hace el comando whatis.

4.1.2.2. Tipee whatis printf. Como podemos ver, existen resultados en más de una sección.

4.1.2.3. ¿Qué es necesario tipear para lograr el objetivo del punto 4.1.1.5? (revea el punto 4.1.1.1 si es necesario y busque como indicar en qué sección buscar).

4.1.2.4. Al escribir por ejemplo "cd /bin" nos desplazamos hacia el directorio /bin. ¿Pero que hace exactamente "cd"? Tipee "whatis cd".

4.1.3. whereis

4.1.3.1. Investigue que hace el comando whereis.

4.1.3.2. Tipee "whereis ls", "whereis socket" y "whereis printf" .

4.1.3.3. Del punto 1.2.4 seguimos sin tener una descripción formal de "cd". Tipee "whereis cd" y vea que sucede.

4.1.4. help

4.1.4.1. Investigue que hace el comando help tipeando "help help".

4.1.4.2. Tipee "help cd".

4.1.4.3. ¡Finalmente logramos el objetivo! Dados los resultados del punto 4.1.2.4, 4.1.3.2, 4.1.3.3 y este punto, ¿Qué diferencia existe (no funcionalmente hablando) entre "cd" y por ejemplo "ls"? ¿Cual de estos dos comandos es un "built-in command"?

4.1.5. apropos

4.1.5.1. Investigue que hace el comando apropos.

4.1.5.2. Supongamos que estamos buscando una función de C que se encarga de suspender la ejecución del proceso que la llama por un tiempo determinado. Tipee "apropos time" y vea si encuentra una función que cumpla con tales características.

4.1.6. info: es un programa para leer documentación, entre la cual se incluyen tutoriales para efectuar distintas tareas en Linux. Este se compone de una estructura del tipo árbol, dividido en nodos de información. Cada nodo describe un tópico específico con un determinado nivel de detalle, el mismo se encuentra señalado con un * (asterisco) y se puede acceder a él posicionando el cursor encima y teclando <enter>.

4.1.6.1. Investigue un poco más el comando info tipeando "man info".

4.1.6.2. Ingrese al programa info tipeando "info".

4.1.6.3. Para navegar entre los nodos de información, algunas opciones son:

4.1.6.3.1. u: desplaza al nodo superior.

4.1.6.3.2. n: desplaza al nodo siguiente.

4.1.6.3.3. p: desplaza al nodo previo.

4.2. Teclado / Terminales

4.2.1. ¿Qué sucede si tecleo cat /e <tab> p <tab>? (donde tab es la tecla tabulación). Presione <tab> nuevamente ¿Qué pasó ahora?

4.2.2. ¿Qué sucede si tecleo cat /e <tab> pas <tab>?

4.2.3. En este punto analizaremos las distintas terminales que hay en un sistema GNU/Linux. Ejecute los siguientes comandos e indique cuál fue el resultado:

4.2.3.1. who

4.2.3.2. Presione la tecla <alt>, y sin soltarla presione cualquiera de las teclas de función. En la pantalla debería aparecer el login del sistema, de lo contrario, ejecute el paso nuevamente presionando otra tecla de función. Si ya tiene el login del sistema vuelva a loguearse.

4.2.3.3. Ejecute nuevamente el comando who. ¿Qué diferencias encuentra con la primera vez que lo ejecutó?

4.2.3.4. Ejecute el comando whoami ¿qué muestra?, ¿Qué diferencias tiene con el comando ejecutado en el punto anterior?

4.2.3.5. Repita el paso 2.3.2 y el 2.3.3 hasta que no encuentre ninguna sesión para abrir.

4.2.3.6. Una vez terminado el punto anterior, Ud. se encontrará sesionado en el sistema como mínimo seis veces. Lo que acaba de hacer es abrir seis terminales virtuales (que podrían ser usadas por distintos usuarios, con diferentes perfiles), en la misma máquina. Así como existen terminales virtuales dentro del mismo equipo, si Ud. cuenta con una red, o con terminales tipo serie, podría abrir tantas sesiones de trabajo como Ud. quiera o necesite.

4.2.3.7. "Todo en Linux es un archivo", y las terminales no son la excepción. Cada Terminal está representada por un archivo llamado ttyx donde x es un número de Terminal, y dichos archivos se encuentran dentro del directorio /dev.

4.2.3.8. Tipee "whatis echo" para saber rápidamente qué hace el comando echo. Luego asegúrese de estar logueado en la 1er y 2da Terminal, y desde la 1er Terminal tipee "echo hola! > tty2". ¿Qué pasó? (más adelante aprenderá en detalle el uso del '>').

4.3. Sistema

4.3.1. Investigue los comandos:

4.3.1.1. halt

4.3.1.2. reboot

4.4. Usuarios

4.4.1. ¿Qué es la cuenta de superusuario (root) y para qué se utiliza? (probablemente tenga que buscarlo en internet).

4.4.2. Ingresar al sistema como superusuario (root), y realizar los siguientes pasos (éste punto no puede ser realizado en el laboratorio):

4.4.3. adduser/addgroup

4.4.3.1. Investigue que hace el comando adduser/addgroup .

4.4.3.2. Cree un nuevo usuario, cree un nuevo grupo, y agregue el usuario a ese grupo.

4.4.4. deluser/delgroup

4.4.4.1. Investigue que hace el comando deluser/delgroup.

4.4.4.2. Borre el usuario creado anteriormente (incluyendo el borrado de su directorio en home y todos sus archivos).

4.4.4.3. ¿Cómo haría para que se borre el directorio home del usuario y todos sus archivos sin tener que mandarlo por parámetro al comando deluser?

4.4.5. Investigue como hacer para saber todos los grupos a los que pertenece un usuario.

4.5. Archivos

4.5.1. ¿Qué hacen los siguientes comandos?

4.5.1.1. cp

4.5.1.2. mv

4.5.1.3. rm

4.5.1.4. scp

4.5.1.5. telnet

4.5.1.6. ssh

4.5.1.7. touch

4.5.2. A la hora de referirse a archivos, se puede usar tanto su dirección relativa (al directorio en el que se encuentra situado) o absoluta. Sitúese como root dentro del directorio /root. Luego copie el archivo .bashrc a la ruta absoluta /var/.bashrc. Ahora, mueva ese archivo desde esa dirección hasta /home/.bashrc sin desplazarse del directorio inicial (/root).

4.6. Permisos

4.6.1. Cree un archivo tipeando "ls > archivo".

4.6.2. Tipee ls –l en dicho directorio: los primeros 10 caracteres corresponden a los permisos. Investigue como se estructuran los permisos de un archivo (puede tipear info y luego ir a la sección de "permisos de archivo" o "file permissions").

4.6.3. chmod/chown

4.6.3.1. Investigue que hacen esos comandos.

4.6.3.2. Haga que el archivo "archivo" creado anteriormente pueda ser modificado por cualquier usuario.

4.6.3.3. Compruebe que logró el punto anterior logueandose en otra Terminal con otro usuario y modificando dicho archivo (tipeando nuevamente "ls > archivo").

4.6.3.4. Loguéese con el usuario original y quite los todos los permisos del archivo (lectura, escritura y ejecución) a todos los usuarios distintos del dueño y de los que pertenecen al mismo grupo. Luego, haga que el nuevo dueño del archivo sea el otro usuario.

4.6.3.5. ¿Cómo haría para volver a poseer dicho archivo sin loguearse con el nuevo dueño del archivo?

4.6.3.6. Investigue qué es el "sticky bit" o "bit pegajoso" (busque en "man chmod").

4.6.3.7. Haga que cualquier usuario distinto del root pueda ejecutar el comando "mount /media/cdrom0" para montar el dispositivo cdrom (si no tiene cdrom puede usar el floppy).

4.6.3.8. Investigue como aplica la estructura de los permisos a los directorios.

4.6.3.9. Loguéese como root en otra Terminal y cree un directorio tipeando "mkdir /undir".

4.6.3.10. Haga que cualquier usuario tenga todo tipo de permisos sobre ese directorio.

4.6.3.11. Deshaga lo que acaba de hacer, y cree el subdirectorio "subdir" dentro de "/undir".

4.6.3.12. Investigue como cambiar los permisos de manera recursiva sobre /undir para que todos sus archivos, subdirectorios y archivos dentro de los subdirectorios se vean afectados.

4.7. Directorios

4.7.1. ¿Para qué se usa el comando cd?. Ejecute las siguientes variantes de cd y observe cuál fue el resultado obtenido:

4.7.1.1. cd /

4.7.1.2. cd

4.7.1.3. cd /etc

4.7.1.4. cd .

4.7.1.5. cd ..

4.7.2. mkdir, rmdir, rm (nuevamente)

4.7.2.1. Investigue dichos comandos.

4.7.2.2. Borre un directorio que no se encuentra vacío.

4.7.2.3. Dentro de /home/<usuario> cree el directorio undir.

4.7.2.4. Ingrese a dicho directorio, y tipee lo siguiente para crear muchos archivos "while (true) do ps > $RANDOM.text; done;". Tipee ctrl.+c luego de 5 seg para finalizar el comando. Luego tipee ls para corroborar la creación de los archivos.

4.7.2.5. Tipee "rm *" e investigue que pasó.

4.8. Filtros

4.8.1. ¿Cuál es la diferencia de los comandos more, less y cat?. Cree un archivo de texto tipeando "ps –fea > texto" y visualícelo con los distintos comandos.

4.8.1.1. Investigue como buscar cadenas de texto cuando se visualiza un archivo con less. ¿Y como se hace para repetir la búsqueda? ¿Y para repetir la búsqueda hacia atrás? Esto le servirá cuando lea páginas del man! (ya que se leen mediante el less).

4.8.2. ¿Cuál es la diferencia entre tail y head?. ¿Qué hace la opción –f del comando tail?

4.8.2.1. Loguéese en una Terminal y tipee "echo > a.txt" para crear el archivo "a.txt". Luego tipee "tail –f a.txt".

4.8.2.2. Desde otra Terminal, tipee "while (true) do date >> a.txt; sleep 2; done;".

4.8.2.3. Vuelva a la terminal anterior y vea lo que sucede.

4.8.2.4. No se olvide de finalizar el comando de la 2da Terminal! (con ctrl.+c)

4.8.3. ¿Qué es lo que realiza el comando sort?

4.8.4. ¿Qué es lo que realiza el comando uniq?

4.8.5. grep

4.8.5.1. ¿Para qué sirve?

4.8.5.2. Busque en el archivo "texto" todas las líneas que contengan la palabra "root".

4.9. Redireccionamiento

4.9.1. Antes de que un comando sea ejecutado, su entrada/salida estándar pueden ser redireccionados usando una notación especial del shell. Investíguelo tipeando "man bash" y llegando luego a la sección "REDIRECTIONS".

4.9.1.1. Redireccionando la salida estándar.

4.9.1.1.1. Ejecute el comando "ps -fea" y redirija su salida a un archivo llamado "salida.txt".

4.9.1.1.2. Ídem punto anterior, pero que se agregue la salida del comando al final del archivo.

4.9.1.2. Redireccionando la entrada estándar.

4.9.1.2.1. El comando "grep cadena archivo" imprime las líneas de archivo que contengan "cadena". Investigue como hacer para lograr el mismo resultado sin especificarle a grep un archivo (investigue si es necesario qué hace grep cuando no se le especifica un archivo).

4.10. Pipelines

4.10.1. El carácter | (pipe) se usa para conectar la salida estándar de un comando con la entrada estándar de otro. Investíguelo tipeando "man bash" y llegando luego a la sección "Pipelines".

4.10.2. Haciendo uso de ps y grep, liste todos los procesos del usuario root.

4.10.3. Usando pgrep, liste todos los PIDs (Process Ids) de procesos que tengan "bash" en su comando de ejecución, redirija la salida a un archivo de texto, y repita esto último 2 veces más (agregando al final del archivo). Luego, liste el contenido del archivo de manera ordenada, eliminando las líneas repetidas y almacene dicho listado en un archivo (todo esto en un mismo comando!).

4.11. Vim

4.11.1. Vim es uno de los editores de texto que vienen por defecto instalados en todo sistema Linux.

4.11.2. Tipee "man vim" para investigar un poco sus características.

4.11.3. Para crear un archivo y editarlo con el vim, tipee "vim archivo.txt".

4.11.4. Para comenzar a escribir debe ingresar al modo edición, presionando la tecla a. Escriba un poco de texto y luego salga del modo de edición presionando <ESC>.

4.11.5. Para grabar el archivo, presione :w (estando fuera del modo edición).

4.11.6. Para salir del editor, presione :q (estando fuera del modo edición).

4.11.7. Para profundizar sobre el vim (más adelante, cuando sea necesario) puede tipear "vim". Así entrará a una pantalla desde la cual podrá tipear ":help" y ingresar a la ayuda. También puede tipear vimtutor, un programa que ofrece un tutorial completo del vim.

4.12. Montar Dispositivos / Filesystems

4.12.1. El comando mount sirve para montar en el filesystem actual otros filesystems o dispositivos.

4.12.1.1. Investigue el comando mount.

4.12.1.2. Monte el diskete/cdrom.

4.12.1.3. Desmóntelo.

4.12.1.4. ¿Para qué sirve el archivo /etc/fstab?

5. Segunda Parte: Solaris

Esta segunda parte del práctico pretende introducir al alumno a entorno Solaris. Se presentan los ejemplos de las salidas de dichos comandos en distintos entornos con diferentes configuraciones (Por ejemplo: sistemas con multiples procesadores). El alumno deberá ejecutar dichos comandos y comparar los resultados obtenidos en su entorno. También se deberá investigar los comandos cuando se lo solicite.

5.1. Información del sistema

A continuación se muestran comando útiles que se utilizan para obtener información sobre el sistema.

Configuración del sistema:

```
# prtconf
```

Un ejemplo para saber cuanta memoria RAM dispone el equipo:

```
# prtconf | head
System Configuration:  Sun Microsystems  i86pc
Memory size: 130560 Megabytes
System Peripherals (Software Nodes):

i86pc
    scsi_vhci, instance #0
        disk, instance #0 (driver not attached)
        disk, instance #1 (driver not attached)
        disk, instance #9
    ib, instance #0 (driver not attached)
```

Para una visión general de la máquina

```
$ prtdiag

System Configuration:  Sun Microsystems  sun4u Sun SPARC Enterprise
M4000 Server
System clock frequency: 1012 MHz
Memory size: 65536 Megabytes

==================================  CPUs ==================================
      CPU                 CPU                           Run    L2$    CPU   CPU
LSB   Chip                 ID                           MHz     MB    Impl.
Mask
---   ----  ----------------------------------------    ----   ---    ----- ---
-
 00     0      0,   1,   2,   3                         2150   5.0    6  146
 00     1      8,   9,  10,  11                         2150   5.0    6  146
 00     2     16,  17,  18,  19                         2150   5.0    6  146
 00     3     24,  25,  26,  27                         2150   5.0    6  146
============================  Memory Configuration ========================

       Memory  Available           Memory     DIMM    #
of  Mirror  Interleave
LSB    Group   Size                Status     Size    DIMMs Mode    Factor
---    ------  ------------------  -------    ------  ----- ------- ---------
-
```

```
 00    A        32768MB            okay       2048MB     16 no       8-way
 00    B        32768MB            okay       2048MB     16 no       8-way
========================  IO Cards  ========================
LSB    Name           Model
---    -------------  ------------
 00    scsi           LSI,1064
 00    network        N/A
 00    network        N/A
 00    SUNW,emlxs     LPe11002-S
 00    SUNW,emlxs     LPe11002-S
 00    SUNW,qlc       QLE2462
 00    SUNW,qlc       QLE2462
====================  Hardware Revisions  ====================
System PROM revisions:
----------------------

OBP 4.24.1 2007/03/06 16:55
==================  Environmental Status  ==================

Mode switch is in LOCK mode
```

Acá tenemos lo mismo para un procesador AMD:

```
$ prtdiag

System Configuration:  Sun Microsystems  Sun Fire X4600 M2
BIOS Configuration:  American Megatrends Inc.  080012  04/19/2007
BMC Configuration:  IPMI 1.5 (KCS: Keyboard Controller Style)

==== Processor Sockets ====================================

Version                          Location Tag
-------------------------------  --------------------------
Dual-Core AMD Opteron(tm) Processor 8222 CPU 1
Dual-Core AMD Opteron(tm) Processor 8222 CPU 2
Dual-Core AMD Opteron(tm) Processor 8222 CPU 3
Dual-Core AMD Opteron(tm) Processor 8222 CPU 4
Dual-Core AMD Opteron(tm) Processor 8222 CPU 5
Dual-Core AMD Opteron(tm) Processor 8222 CPU 6
Dual-Core AMD Opteron(tm) Processor 8222 CPU 7
Dual-Core AMD Opteron(tm) Processor 8222 CPU 8

==== Memory Device Sockets ================================

Type    Status Set Device Locator      Bank Locator
------- ------ --- ------------------  --------------------
DDR2    in use 0   DIMM0               BANK0
DDR2    in use 0   DIMM1               BANK1

==== On-Board Devices =====================================
 LSI serial-ATA #1
 Gigabit Ethernet #1
 Gigabit Ethernet #2
 ATI Rage XL VGA

==== Upgradeable Slots ====================================

ID  Status    Type             Description
--- --------- ---------------  ----------------------------
0   in use    PCI-X            PCIX SLOT0
```

```
1   available PCI-X            PCIX SLOT1
2   available other            PCIExp SLOT2
3   available other            PCIExp SLOT3
4   available other            PCIExp SLOT4
5   available other            PCIExp SLOT5
6   available other            PCIExp SLOT6
7   available other            PCIExp SLOT7
```

Con el siguiente comando se puede observar la lista de procesadores que existen en el equipo. Muestra cuales están operativos y cuales no.

```
$ psrinfo –v

Status of virtual processor 0 as of: 02/24/2009 17:07:26
  on-line since 02/20/2009 13:08:07.
  The i386 processor operates at 2660 MHz,
        and has an i387 compatible floating point processor.
Status of virtual processor 1 as of: 02/24/2009 17:07:26
  on-line since 02/20/2009 13:08:14.
  The i386 processor operates at 2660 MHz,
        and has an i387 compatible floating point processor.
Status of virtual processor 2 as of: 02/24/2009 17:07:26
  off-line since 02/20/2009 12:10:46.
```

Para cambiar el estado operacional de un procesador.

```
$ psradm
```

Nota: ¿qué pasa si apago todos los procesadores? No pasa nada, porque nunca se apagan todos. Aunque le pida a psradm que los apague todos siempre queda 1 on-line.

Para controlar quien usa que procesador:

psrset: se pueden armar subgrupos de procesadores para particionar los recursos de cómputo de la máquina.
pbind: se puede asignar un procesador a un proceso o a un thread. (válido cuando existen múltiples procesadores)

Para saber si estoy en una plataforma 32 o 64 bits:

```
# isalist   # ejemplo con sparc
sparcv9+vis2 sparcv9+vis sparcv9 sparcv8plus+vis2 sparcv8plus+vis
sparcv8plus sparcv8 sparcv8-fsmuld sparcv7 sparc

#isalist    # ejemplo con AMD
amd64 pentium_pro+mmx pentium_pro pentium+mmx pentium i486 i386 i86

# isainfo -v
64-bit sparcv9 applications
        fmaf vis2 vis popc
32-bit sparc applications
        fmaf vis2 vis popc v8plus div32 mul32
```

Para investigar: ¿Por qué muestra, estando en una máquina de 64 bits, la información para 32 bits?

Para que se utilizan los siguientes comandos:

cputrack
cpustat

busstat

```
$ uname -a
SunOS fritanga 5.10 Generic_138888-03  sun4u sparc SUNW,SPARC-Enterprise
```

Información sobre los discos rígidos. ¿Qué información le brinda al usuario del sistema?

```
# format
Searching for disks...
Inquiry failed on 4
Inquiry failed on 4
done

AVAILABLE DISK SELECTIONS:
       0. c0t0d0 <DEFAULT cyl 17830 alt 2 hd 255 sec 63>
          /pci@0,0/pci8086,25f8@4/pci108e,286@0/sd@0,0
       1. c0t1d0 <Sun-STK RAID INT-V1.0-409.85GB>
          /pci@0,0/pci8086,25f8@4/pci108e,286@0/sd@1,0
       2. c2t600A0B8000488B3E0000049F499EA4BCd0  <drive type unknown>
          /scsi_vhci/disk@g600a0b8000488b3e0000049f499ea4bc
       3. c2t600A0B8000488F98000004AA499EA44Cd0  <drive type unknown>
          /scsi_vhci/disk@g600a0b8000488f98000004aa499ea44c
Specify disk (enter its number): ^C
```

Información sobre las interfaces de red disponibles.

```
$ ifconfig –a

lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL>  mtu 8232
index 1
        inet 127.0.0.1 netmask ff000000
ce1: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4>  mtu 1500 index 2
        inet 129.157.210.7  netmask ffffff00 broadcast 129.157.210.255
```

5.2. Información sobre los procesos en el sistema

A continuación se presenta información con una descripción detallada de las salidas de los comandos. Mucha de esta información será de gran ayuda para resolver problemas de programación a bajo nivel. Es conveniente que el alumno haga una lectura del mismo.

Qué procesos existen en este momento?

```
$ ps -ef
```

o alguno en particular

```
$ ps -ef | grep fire
canary 25630 25615   0   Feb 06 ?           0:00 /bin/sh
/usr/dist/share/firefox,v2.0.0.16/5.10-bin.sun4/../5.10-lib.sun4/firefo
vperrot 28945 28923   0   Feb 16 pts/151     0:00 /usr/xpg4/bin/sh
/usr/dist/share/firefox,v2.0.0.16/5.10-bin.sun4/firefox
orivat 53264 53253   0   Feb 18 ?           0:00 /bin/sh
/usr/dist/share/firefox,v2.0.0.16/5.10-bin.sun4/../5.10-lib.sun4/firefo
```

Los procesos usan recursos del sistema. Un típico recurso es la CPU.
Cual es el consumo actual de CPU?

```
$ vmstat 5
 kthr      memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr m0 m1 m2 m3   in   sy   cs us sy id
 0 0 0 59318448 2792472 190 1540 40 22 20 0 2 2 0 10 0 1953 22390 7315 20 13 67
 0 0 0 54997656 711496 237 4195 18 0 0 0 0  3  0  1  0 2361 28635 9467 28 25 47
 0 0 0 54989032 695872 340 1808 887 148 91 0 0 11 0 12 0 2319 35107 7943 41 27 31
 0 0 0 54971208 667344 169 1988 29 104 102 0 0 2 0 32 0 2434 26884 8175 50 23 28
 1 0 0 54959864 653736 93 1957 235 126 126 0 0 1 0 22 0 2734 26149 8646 42 23 35
 1 0 0 54950408 641640 157 2459 208 332 268 0 0 1 0 70 0 2857 48758 8191 53 28 18
 1 0 0 55000584 766384 196 3338 365 0 0 0 0 17 0  0  0 3647 52004 11294 52 30 18
 0 0 0 55006888 773096 45 1884 40 0 0 0  0  7  0  0  0 2353 22236 9802 48 16 35
 0 0 0 55008392 774728 37 497 0 0  0  0  0  0  0  5  0 1724 15058 7941 25 10 65
```

Las columnas a la derecha us, sy y id dan el porcentaje de tiempo pasado en user space, systeme space y ocioso (idle). *Ver los conceptos de estar ejecutando en espacio usuario y espacio sistema y de proceso ocioso.*

Las columnas kthr indican los procesos que están en estado "ready", "busy" o "waiting". *Ver concepto de estado de un proceso (listo, bloqueado, running,….).* Por ejemplo los 1s en kthr r, indica que hubo un proceso listo para ejecutar, en espera de la CPU.

Las columnas in, sy y cs indican respectivamente las interrupciones, los system calls y los context switch por segundo. *Ver conceptos: interrupciones, llamadas al sistema y cambio de contexto.*

La visión de uso de la CPU que vimos con vmstat es global a todos los procesadores (*Ver concepto de multiprocesador*). Si por ejemplo tenemos 1 procesador usado a 100% por un proceso con un solo thread pero tengo 4 procesadores, el indicador me dirá que sólo uso 25% de CPU porque los otros 3 estarán en 0% de uso; cuando en realidad ya estoy saturado porque en este caso no puedo ir más rápido (por qué?). Para ver como está repartida la carga entre los procesadores:

```
$ mpstat 5
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0    0   0   41  1327  272 3376   10   14  532    0  3533   95   3   0   2
  1    0   0   20   995    0 3381    8   14  562    0  3546   95   2   0   3
  2    0   0   32  1563    0 3096   25   32  505    0  2902   82   3   0  15
  3    0   0    8  1260    0 3384   14   21  534    0  3206   90   2   0   8
  4    4   0   39  1771    1 2834   11   28  442    0  2850   75   2   0  23
  5    0   0  152  2725    3 1916   11   58  268    0  1684   47   1   0  52
  6    0   0    8  1001    0 2510    7   32  405    0  2342   64   2   0  34
  7    0   0   28   938   24 2260    4   25  350    0  2178   58   2   0  40
  8    0   0  221  1108   84 3263   37   38  477    0  3017   80   2   0  18
  9    0   0   28   987    0 3752   12   14  579    0  3495   94   2   0   4
 10    0   0   16   993    2 3455   13   10  560    0  3397   93   3   0   4
 11    7   0   16   986    0 3523   11    8  571    0  3445   95   2   0   3
 12    0   0   28  1002    0 3417    9    8  552    0  3497   96   2   0   2
 13    0   0   32   811    0 3445    9    7  533    0  3449   95   2   0   3
 14    0   0  121   690    0 2716    7   16  410    0  2691   71   2   0  27
 15    0   0    8   338    0 1217    2   13  175    0  1259   34   1   0  65
```

Aquí vemos por ejemplo, que el procesador 0 esta casi totalmente utilizado (98%) y el procesador 5 está utilizado solo por la mitad (48%). Noten también que, entre otros, hay información por procesador de los cambios de contexto (csw), cambios de contexto involuntarios (icsw), migraciones, mutex (smtx), llamadas al sistema (syscl)...

Un típico indicador de la carga de procesamiento de una máquina se ve con:

```
$ uptime

  5:12pm  up 98 day(s), 22:56,  18 users,  load average: 10.34, 6.57, 6.06
```

Da indicación de la carga que soportó el sistema de procesamiento (load average). Indica un índice de carga en los 1, 5 y 15 últimos minutos. La manera fácil de interpretar estos números sería así: un número igual al de mi cantidad de procesadores sería una utilización del 100%. Menos, sería proporcionalmente menos que el 100%. Un valor mayor, sería un indicador de saturación. En este caso había 16 procesadores. Por ende vemos que la carga fue aumentado desde los últimos 15 minutos (de 6.06 hace 15 minutos, pasó a 6.57 hace 10 minutos y paso a 10.34 hace 1 minuto).

El utilitario que mucha gente usa es el top, para ver el uso de CPU por proceso. El top existe también para Solaris. Sin embargo un utilitario equivalente y más poderoso es:

```
$ prstat

   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 53310 orivat    334M  170M sleep   59    0  13:29:33 3.5% firefox-bin/10
 29003 vperrot   258M  214M run     37    4   6:46:56 3.3% firefox-bin/9
 43415 jedebais  350M  276M sleep   59    0   7:17:32 3.0% firefox-bin/14
 42155 lganne    361M  303M sleep   59    0  87:55:32 2.2% Xsun/1
 12457 sdussud   219M  165M sleep   59    0   0:22:58 2.0% thunderbird-bin/13
 64579 orivat    256M  179M sleep   59    0  36:36:40 1.5% Xsun/1
 47707 pvazquez  423M  268M sleep   59    0   7:04:29 1.5% firefox-bin/10
 46224 ewathele  159M  126M sleep   59    0   0:12:35 1.5% thunderbird-bin/12
  5034 pvazquez  299M  227M sleep   55    0   7:21:38 1.5% Xsun/1
 30099 rg148669  309M  249M cpu2    59    0   1:58:17 1.1% firefox-bin/8
 47604 root       40M   32M sleep   59    0  96:09:31 0.9% esd/2
 40198 bbaret    230M  177M sleep   59    0   0:18:41 0.9% thunderbird-bin/15
 28741 rg148669  182M  144M sleep   59    0   7:12:20 0.9% Xsun/1
 45714 lganne    577M  507M sleep   59    0   6:26:55 0.4% gnome-terminal/2
 39138 ewathele  214M  175M sleep   59    0   8:26:25 0.4% Xsun/1
  4923 pvazquez  129M   42M sleep   49    0   0:24:53 0.4% gnome-terminal/2
 24571 em231301  253M  191M sleep   37    4   0:10:15 0.4% acroread/1
 38362 orivat    249M  160M sleep   49    0   0:27:10 0.3% acroread/1
 14757 em231301  219M  140M sleep   37    4   0:08:19 0.2% soffice.bin/7
 16259 lganne     82M   66M sleep   49    0   1:09:25 0.2% nedit/1
 27482 pvazquez 4768K 4296K cpu2    19    0   0:00:00 0.2% prstat/1
  5541 pvazquez   94M   39M sleep   59    0   0:37:04 0.2% metacity/1
Total: 1581 processes, 2824 lwps, load averages: 4.61, 6.68, 6.50
```

Permite ver que procesos, los usuarios, etc. que usan la CPU y otros recursos como la memoria. Noten que al final del display se informa del número de procesos el número de threads (lwps) y el load average (como el del comando uptime). Un ejemplo más poderoso de prstat:

```
$ prstat -m
   PID USERNAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROCESS/NLWP
  1173 pvazquez 0.0 0.0 0.0 0.0 0.0 0.0 100 0.0  29   0 261   0 prstat/1
  1085 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0  15   0  10   0 tail/1
   317 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 cron/1
   465 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 automountd/2
   398 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 smcboot/1
   299 daemon   0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 nfs4cbd/2
   397 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 smcboot/1
   326 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 utmpd/1
   312 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 sac/1
   132 daemon   0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 kcfd/3
   327 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 ttymon/1
   135 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 powerd/3
   311 daemon   0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 nfsmapid/4
   276 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   3   0   1   0 in.routed/1
   123 root     0.0 0.0 0.0 0.0 0.0  25  75 0.0   0   0   0   0 picld/4
   148 root     0.0 0.0 0.0 0.0 0.0  50  50 0.0   0   0   0   0 devfsadm/6
   286 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 keyserv/3
   151 root     0.0 0.0 0.0 0.0 0.0 2.9  97 0.0  29   0 109   0 nscd/34
   300 daemon   0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 lockd/2
```

```
   328 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 ttymon/1
   285 daemon   0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 statd/1
   483 root     0.0 0.0 0.0 0.0 0.0  62  38 0.0   0   0   0   0 syslogd/13
   120 root     0.0 0.0 0.0 0.0 0.0  71  29 0.0   0   0   0   0 syseventd/14
   247 daemon   0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   0   0   0   0 rpcbind/1
     9 root     0.0 0.0 0.0 0.0 0.0  12  87 0.0   0   0   0   0
svc.configd/16
     7 root     0.0 0.0 0.0 0.0 0.0  38  62 0.0   0   0   0   0 svc.startd/13
Total: 55 processes, 186 lwps, load averages: 1.38, 2.28, 1.80
```

Da la utilización pero también los micro-estados de un proceso (*Ver concepto de estados de un proceso*): que porcentaje de tiempo pasa en tiempo usuario, en locks (LCK), durmiendo (SLP), sirviendo señales (SIG), cambios de contexto voluntarios como cuando pide una entrada y salida, e involuntarios como cuando se le termino el time slice en el round robin y se lo pasa a la cola de listos (VCX, ICX), esperando la CPU en la cola de listos (LAT), etc....

O también se puede ver lo mismo pero a nivel de los threads:

```
$  prstat -mL
   PID USERNAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROCESS/LWPID
  1196 root      16  13 0.0 0.0 0.0 0.9  70 0.1  1K  18 24K   0 java/2
  1196 root     0.4 0.0 0.0 0.0 0.0 100 0.0 0.0  24   0  25   0 java/10
  1196 root     0.4 0.0 0.0 0.0 0.0 100 0.0 0.0 722   0 724   0 java/3
  1196 root     0.4 0.0 0.0 0.0 0.0  99 0.1 0.0  1K   0  1K   0 java/15
  1196 root     0.4 0.0 0.0 0.0 0.0  99 0.1 0.0  1K   0  1K   0 java/12
  1196 root     0.4 0.0 0.0 0.0 0.0  99 0.1 0.0  1K   0  1K   0 java/7
  1196 root     0.4 0.0 0.0 0.0 0.0  99 0.1 0.0  1K   0  1K   0 java/9
  1196 root     0.4 0.0 0.0 0.0 0.0  99 0.1 0.0  1K   0  1K   0 java/8
  1196 root     0.4 0.0 0.0 0.0 0.0 100 0.0 0.0 612   0 618   0 java/5
  1196 root     0.4 0.0 0.0 0.0 0.0  99 0.1 0.0  1K   0  1K   0 java/13
  1196 root     0.4 0.0 0.0 0.0 0.0 100 0.0 0.0 609   0 610   0 java/6
  1196 root     0.4 0.1 0.0 0.0 0.0  99 0.1 0.0  1K   0  1K   0 java/14
  1196 root     0.4 0.0 0.0 0.0 0.0 100 0.0 0.0  1K   0  1K   0 java/11
  1196 root     0.4 0.0 0.0 0.0 0.0  99 0.1 0.0  1K   0  1K   0 java/4
  1196 root     0.3 0.0 0.0 0.0 0.0 100 0.0 0.0  21   0  32   0 java/21
  1196 root     0.1 0.0 0.0 0.0 0.0 100 0.0 0.0   2   0   2   0 java/27
  1202 root     0.1 0.0 0.0 0.0 0.0 0.0 100 0.0  15   4 700   0 vmstat/1
  1203 root     0.1 0.0 0.0 0.0 0.0 0.0 100 0.0   5   0 815   5 iostat/1
  1208 root     0.0 0.1 0.0 0.0 0.0 0.0 100 0.0  29   0 303   0 prstat/1
  1196 root     0.1 0.0 0.0 0.0 0.0 0.9  99 0.0  69   0  24   0 java/30
  1196 root     0.1 0.0 0.0 0.0 0.0 100 0.0 0.0   1   0   1   0 java/26
  1196 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0 151   0  88   0 java/29
  1196 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   4   1  27   0 java/31
  1196 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   7   0  27   0 java/34
  1196 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   5   0  26   0 java/33
  1006 gbellato 0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   5   0 115   0 sshd/1
Total: 60 processes, 235 lwps, load averages: 0.50, 1.33, 1.50
```

Acá vemos por ejemplo que el thread 2 del proceso java es el que más CPU consume (29%). *Ver concepto de thread*. Tambien vemos que tuvo 18 cambios de contexto involuntario; lo que explicaria el 0.1 en LAT.

Los procesos son las entidades ejecutantes en un sistema. Estos ejecutan en espacio usuario y en espacio del sistema cuando hacen una llamada al sistema. Estas suelen ser caras. Por eso cuando con uno de los utilitarios anteriores vemos que hay muchas llamadas al sistema se puede uno preguntar:
Quién esta haciendo tantos systems calls? (*Ver todos los conceptos que aparecen en el párrafo*)

```
# dtrace -n 'syscall:::entry {@[execname] = count(); }'
dtrace: description 'syscall:::entry ' matched 229 probes
^C
```

fmd 1

```
  nscd                                                              1
  svc.configd                                                       1
  svc.startd                                                        1
  sendmail                                                         10
  sshd                                                            114
  dtrace                                                          420
  vmstat                                                          564
  iostat                                                          656
  java                                                          25846
```

Qué estamos haciendo con dtrace? Acá le decimos a dtrace, que sume 1 (count()) a un contador. Este contador esta en un array de contadores (un array de enteros) que esta indexado por nombre de programa (`{@[execname]`). O sea que en la posición [iostat], por ejemplo, haremos +1. La palabra execname es una palabre clave de dtrace que indica el nombre del programa que esta inspeccionando. Cuando suma 1 en este programa? Muy simple, cada vez que entra en un system call (`syscall:::entry`). O sea, que cada vez que un programa hace un system call (entra a un system call) se suma 1 en un contador, identificado por el nombre del programa que llama al system call. De esta manera tenemos cuantos llamados al sistema hizo cada programa. Y todo esto en una línea de comando.

Lo dejamos ejecutar unos segundos y lo cortamos con ^C y vemos el resultado. Ahaa!! Parece que es el proceso java el que más systems calls está haciendo (40 veces más que cualquier otro). Vayamos más a fondo en la investigación y tratemos de ver que cosa en este programa java es el que hace tantos system calls:

```
# dtrace -n 'syscall:::entry /execname == "java"/ {@[probefunc] = count(); }'
dtrace: description 'syscall:::entry ' matched 229 probes
^C

  lwp_cond_signal                                                   1
  times                                                             1
  ioctl                                                             2
  close                                                             3
  fcntl                                                             3
  fsat                                                              3
  fstat                                                             3
  mprotect                                                          3
  getdents                                                          6
  lwp_park                                                          6
  write                                                             6
  lwp_cond_broadcast                                               14
  lwp_cond_wait                                                    20
  stat                                                             45
  pollsys                                                          77
  lwp_mutex_timedlock                                          103863
  lwp_mutex_wakeup                                             107898
```

Acá con dtrace hacemos lo mismo que antes, salvo que filtramos para sólo ver al proceso java (`/execname == "java"/`). Y esta vez el array lo indexamos por nombre de función (`@[probefunc]`), porque lo que nos interesa es saber dentro de java que es lo que produce tantos system calls.

De vuelta lo dejamos correr un poquito y lo cortamos. Parece que quien más genera system calls son los llamados a mutex por parte de los threads del programa java (lwp: proceso liviano o thread reconocido por el SO). (*Ver concepto de mutex y de porque esto es un llamado al sistema*).

Noten como, con dos comandos, pasamos a saber exactamente de donde viene el consumo mayor de llamadas al sistema.

Ya vimos

```
$ prstat
   PID USERNAME  SIZE   RSS STATE   PRI NICE      TIME  CPU PROCESS/NLWP
  1196 root       98G   27G cpu9     0    0   0:15:18 6.7% java/45
  1202 root     2436K 1676K sleep   59    0   0:00:00 0.0% vmstat/1
  1203 root     2576K 1792K sleep   59    0   0:00:01 0.0% iostat/1
   465 root     4484K 1132K sleep   59    0   0:00:00 0.0% automountd/2
   398 root     1736K  692K sleep   59    0   0:00:00 0.0% smcboot/1
   299 daemon   2124K 1340K sleep   60  -20   0:00:00 0.0% nfs4cbd/2
   397 root     1736K  692K sleep   59    0   0:00:00 0.0% smcboot/1
   326 root     1108K  640K sleep   59    0   0:00:00 0.0% utmpd/1
   312 root     1748K  976K sleep   59    0   0:00:00 0.0% sac/1
   132 daemon   3956K 2084K sleep   59    0   0:00:00 0.0% kcfd/3
   327 root     2128K 1320K sleep   59    0   0:00:00 0.0% ttymon/1
   135 root     1500K 1052K sleep   59    0   0:00:00 0.0% powerd/3
   311 daemon   2432K 1172K sleep   59    0   0:00:00 0.0% nfsmapid/4
   276 root     2164K 1152K sleep   59    0   0:00:00 0.0% in.routed/1
   123 root     3904K 2892K sleep   59    0   0:00:00 0.0% picld/4
   148 root     3068K 1868K sleep   59    0   0:00:00 0.0% devfsadm/6
   286 root     2264K 1188K sleep   59    0   0:00:00 0.0% keyserv/3
   151 root     7992K 4180K sleep   59    0   0:00:00 0.0% nscd/34
   300 daemon   2096K 1352K sleep   60  -20   0:00:00 0.0% lockd/2
   328 root     2036K 1204K sleep   59    0   0:00:00 0.0% ttymon/1
   285 daemon   2448K 1604K sleep   59    0   0:00:00 0.0% statd/1
   483 root     3704K 1720K sleep   59    0   0:00:00 0.0% syslogd/13
   120 root     2212K 1312K sleep   59    0   0:00:00 0.0% syseventd/14
   247 daemon   2596K 1280K sleep   59    0   0:00:00 0.0% rpcbind/1
     9 root       11M 9568K sleep   59    0   0:00:04 0.0% svc.configd/16
     7 root       15M   13M sleep   59    0   0:00:02 0.0% svc.startd/13
Total: 63 processes, 238 lwps, load averages: 1.31, 1.56, 1.57
```

Que da información sobre los procesos que corren en este memento. Da información sobre el usuario que los lanzó, el pid, el tamaño del proceso, el tamaño que está ocupando realmente en memoria física (RSS) en ese momento (*Ver conceptos de tamaño en memoria virtual y tamaño en memoria física*), el estado (runing, ready, etc.), hace cuanto que corre, el uso de CPU, y muchas mas cosas dependiendo de las opciones.

Por ejemplo el proceso java (primera línea) usa un espacio virtual de memoria de 98G. Pero en memoria física solo esta usando 27G. En este momento esta corriendo en la décima CPU (CPU 9), su pid es 1196 y el usuario root es quien lo lanzó. Hace 15 minutos y 19 segundos que ejecuta y anda usando un 6.7% de CPU. Este proceso esta compuesto de 45 threads (java/45). El resto de los procesos en este display no hacen nada.

Como dijimos, en la mayoría de los casos un usuario lanza un proceso. Pero otro caso muy frecuente es el de un proceso que instancia a otro proceso. Para ver la arborescencia de forkeo de un proceso (*Ver concepto de fork y join*):

```
$ ptree 25630
25566 /usr/dist/pkgs/5bin.sun4/cam firefox -no-remote -P cb_test http://lo
  25615 /usr/xpg4/bin/sh /usr/dist/share/firefox,v2.0.0.16/firefox -no-re
    25630 /bin/sh /usr/dist/share/firefox,v2.0.0.16/../5.10-lib.sun4/firefo
      25663 /bin/sh /usr/dist/share/firefox,v2.0.0.16/../5.10-lib.sun4/run-mo
        25669 /usr/dist/share/firefox,v2.0.0.16/../5.10-lib.sun4/firefox-bin -n
```

Hay varios comandos p* que son muy útiles vean el man page.

El más original quizás sea preap que permite matar a los zombis (*Ver concepto de zombi*. En Solaris a un zombi se le dice defunct). Pasamos años los usuarios de unix esperando este comando.

Algunoscomandosparaintrospección deprocesosenejecución:

```
# pstack 1226
1226:   /usr/lib/ssh/sshd
 fec64727 pollsys  (8047340, 4, 0, 0)
 fec12f5a pselect  (c, 80b80b8, 80c20b8, fec8f180, 0, 0) + 18e
 fec13250 select   (c, 80b80b8, 80c20b8, 0, 0) + 82
 0806c50c ???????? (8047460, 8047464, 804746c, 8047468, 0)
 0806d369 server_loop2 (80b6400, 8047488, 806dec9, 80b6400, 8047ddc, 805dbfb) + c1
 080712c3 ???????? (80b6400)
 0806dec9 do_authenticated (80b6400, 8047e40, 8047dc4, feffa818, 62696c2f, 62696c2f) +
38
 0805dbfb main     (1, 8047e08, 8047e10) + 1006
 0805baf2 ???????? (1, 8047ea8, 0, 8047eba, 8047ed5, 8047eee)
```

Provee una instantánea del stack del proceso en ese momento (*Ver concepto de la composición de un proceso: stack, heap,…*). En este caso podemos ver (se leen los llamados de abajo hacia arriba: es un stack o sea una pila) el main, llama a una autenticación (el proceso es ssh) que entra en un loop donde se queda en select pooleando por la llegada de nuevos datos.

Otro recurso típico que usan los procesos es el sistema de archivos (*ver concepto de file system*). Que archivos (o file descriptors. *Ver concepto de file descriptor*) están usados por un proceso?:

```
# ps -ef | grep java
    root  1196  1195   0 17:21:09 pts/2      38:44 /usr/jdk/instances/jdk1.6.0/bin/amd64/java -
server -Xms96G -Xmx96G -XX:NewSize=
    root  1350  1206   0 17:50:14 pts/4       0:00 grep java
# pfiles 1196
1196:   /usr/jdk/instances/jdk1.6.0/bin/amd64/java -server -Xms96G -Xmx96G -XX
  Current rlimit: 65536 file descriptors
   0: S_IFCHR mode:0620 dev:274,0 ino:12582920 uid:18317 gid:7 rdev:24,2
      O_RDWR|O_NOCTTY|O_LARGEFILE
      /devices/pseudo/pts@0:2
[...]

   5: S_IFREG mode:0644 dev:61,448 ino:312972 uid:0 gid:2 size:49734066
      O_RDONLY|O_LARGEFILE FD_CLOEXEC
      /usr/jdk/instances/jdk1.6.0/jre/lib/rt.jar
   6: S_IFREG mode:0644 dev:61,448 ino:75761 uid:0 gid:2 size:563455
      O_RDONLY|O_LARGEFILE FD_CLOEXEC
      /usr/jdk/packages/javax.help-2.0/lib/jhall.jar
[...]

  14: S_IFSOCK mode:0666 dev:280,0 ino:58939 uid:0 gid:0 size:0
      O_RDWR
        SOCK_STREAM
        SO_SNDBUF(16384),SO_RCVBUF(5120)
        sockname: AF_UNIX
  15: S_IFCHR mode:0644 dev:274,0 ino:78118918 uid:0 gid:3 rdev:149,1
      O_RDONLY|O_LARGEFILE
      /devices/pseudo/random@0:urandom
  16: S_IFREG mode:0644 dev:61,448 ino:312961 uid:0 gid:2 size:88256
      O_RDONLY|O_LARGEFILE FD_CLOEXEC
      /usr/jdk/instances/jdk1.6.0/jre/lib/jce.jar
[...]

 137: S_IFREG mode:0644 dev:181,65538 ino:50791 uid:0 gid:0 size:49996958
      O_RDONLY|O_LARGEFILE
      /ds/ms/OpenDS-1.3.0/db/userRoot/000034bc.jdb
 138: S_IFREG mode:0644 dev:181,65538 ino:50792 uid:0 gid:0 size:49996015
      O_RDONLY|O_LARGEFILE
      /ds/ms/OpenDS-1.3.0/db/userRoot/000034bd.jdb
```

Noten que no todos los file descriptors están en relación a un archivo. Por ejemplo el numero 137 está en relación a un archivo (`000034bd.jdb`). Pero el 14 está en relación a un `SOCK_STREAM` o sea a un socket de protocolo TCP.

Cuanto tarda un proceso en ejecutar?

```
$ ptime sleep 2

real        2.028
user        0.001
sys         0.005
```

Uno de los grandes conceptos en relación a los procesos es IPC (*Ver Inter. Process communication*). Una de las formas de comunicación son las señales (signals). Un proceso puede mapear las señales; o sea, que a la llegada de una señal específica, se le relaciona un pedazo de código (un signal handler).

Como mapeó los signals un proceso?

```
# psig 1196
1196:   /usr/jdk/instances/jdk1.6.0/bin/amd64/java -server -Xms96G -Xmx96G -
XX
HUP     caught  sigacthandler    RESTART
HUP,INT,QUIT,ILL,TRAP,ABRT,EMT,FPE,BUS,SEGV,SYS,PIPE,ALRM,TERM,USR1,USR2,CLD,
PWR,WINCH,URG,POLL,TSTP,CONT,TTIN,TTOU,VTALRM,PROF,XCPU,XFSZ,WAITING,LWP,FREE
ZE,THAW,CANCEL,LOST,XRES,JVM1,JVM2,RTMIN,RTMIN+1,RTMIN+2,RTMIN+3,RTMAX-
3,RTMAX-2,RTMAX-1,RTMAX
INT     caught  sigacthandler    RESTART
HUP,INT,QUIT,ILL,TRAP,ABRT,EMT,FPE,BUS,SEGV,SYS,PIPE,ALRM,TERM,USR1,USR2,CLD,
PWR,WINCH,URG,POLL,TSTP,CONT,TTIN,TTOU,VTALRM,PROF,XCPU,XFSZ,WAITING,LWP,FREE
ZE,THAW,CANCEL,LOST,XRES,JVM1,JVM2,RTMIN,RTMIN+1,RTMIN+2,RTMIN+3,RTMAX-
3,RTMAX-2,RTMAX-1,RTMAX
QUIT    caught  sigacthandler    RESTART
HUP,INT,QUIT,ILL,TRAP,ABRT,EMT,FPE,BUS,SEGV,SYS,PIPE,ALRM,TERM,USR1,USR2,CLD,
PWR,WINCH,URG,POLL,TSTP,CONT,TTIN,TTOU,VTALRM,PROF,XCPU,XFSZ,WAITING,LWP,FREE
ZE,THAW,CANCEL,LOST,XRES,JVM1,JVM2,RTMIN,RTMIN+1,RTMIN+2,RTMIN+3,RTMAX-
3,RTMAX-2,RTMAX-1,RTMAX
ILL     caught  sigacthandler    RESTART,SIGINFO
HUP,INT,QUIT,ILL,TRAP,ABRT,EMT,FPE,BUS,SEGV,SYS,PIPE,ALRM,TERM,USR1,USR2,CLD,
PWR,WINCH,URG,POLL,TSTP,CONT,TTIN,TTOU,VTALRM,PROF,XCPU,XFSZ,WAITING,LWP,FREE
ZE,THAW,CANCEL,LOST,XRES,JVM1,JVM2,RTMIN,RTMIN+1,RTMIN+2,RTMIN+3,RTMAX-
3,RTMAX-2,RTMAX-1,RTMAX
TRAP    default
ABRT    default
EMT     default
```

A qué librerías esta linkeado un proceso?

```
# echo $$
1206
# ps -ef | grep 1206
    root  1206  1205   0 17:22:20 pts/4       0:00 bash
    root  1399  1206   0 17:56:26 pts/4       0:00 grep 1206
    root  1398  1206   0 17:56:26 pts/4       0:00 ps -ef
# pldd $$
1206:   bash
/lib/libcurses.so.1
/lib/libsocket.so.1
/lib/libnsl.so.1
/lib/libdl.so.1
```

```
/lib/libc.so.1
/usr/lib/locale/en_US.ISO8859-1/en_US.ISO8859-1.so.3
```

Que esta haciendo un proceso en este momento?

```
# ps -ef | grep bash
pvazquez  4815  4761   0 11:29:53 pts/5       0:00 bash
    root  1206  1205   0 17:22:20 pts/4       0:00 bash
    root  4824  1206   0 11:31:12 pts/4       0:00 grep bash
# truss -p 4815
read(0, 0x0804639C, 1)          (sleeping...)
read(0, " p", 1)                                = 1
write(2, " p", 1)                               = 1
read(0, " s", 1)                                = 1
write(2, " s", 1)                               = 1
read(0, "\r", 1)                                = 1
write(2, "\n", 1)                               = 1
time()                                          = 1235558445
stat64(".", 0x08046A90)                         = 0
stat64(".", 0x080469B0)                         = 0
stat64("./ps", 0x080469B0)                      Err#2 ENOENT
stat64("/home/pvazquez/mio/usr/local/bin/ps", 0x080469B0) Err#2 ENOENT
stat64("/home/pvazquez/bin/ps", 0x080469B0)     Err#2 ENOENT
stat64("/usr/openwin/bin/ps", 0x080469B0)       Err#2 ENOENT
stat64("/usr/lang/WorkShop_latest/bin/ps", 0x080469B0) Err#2 ENOENT
stat64("/usr/lang/JAVA/jdk1.5/bin/ps", 0x080469B0) Err#2 ENOENT
stat64("/usr/bin/ps", 0x080469B0)               = 0
stat64("/usr/bin/ps", 0x080469D0)               = 0
lwp_sigmask(SIG_SETMASK, 0x00020002, 0x00000000) = 0xFFBFFEFF [0x0000FFFF]
fork1()                                         = 4864
write(2, " / o p t / c s w  "., 17)           = 17
read(0, 0x0804639C, 1)          (sleeping...)
```

Inspeccionamos el shell bash (pid=4815): está en espera de leer algo (read en el file escriptor 0 que es el stardard input, o sea el teclado) , lee el carácter 'p' y lo escribe (escribe en el file descriptor 2 que es el standard output, o sea la pantalla), lee el carácter 's' y lo escribe en la pantalla (o sea que alguien escribió 'ps' en la terminal). Llega el enter (\r\n). Toma la hora. Se fija si existe el comando 'ps' en varios lugares con stat64 (lugares que corresponden a la variable de entorno PATH). Lo encuentra en (retorno 0) en '/usr/bin/ps'. Mapea señales y forkea un proceso para que ejecute el comando 'ps'. Escribe el prompt que consiste en el working directory '/opt/csw', y vuelve a esperar una lectura.

La herramienta truss es extremadamente poderosa para debuguear o ver lo que hace un proceso que está corriendo.

El mismo comando 'ps' en la misma terminal, le pedimos a dtrace la distribución (cuantos pedidos de cada una) de las llamadas que se hacen:

```
# dtrace   -n 'pid$target:::entry  { @[probefunc] = count(); }' -p 4815
dtrace: description 'pid$target:::entry  ' matched 6783 probes
^C
  ___errno                                                        1
  __cerror                                                        1
  __fork1                                                         1
  _flsbuf                                                         1
  _waitpid                                                        1
  sh_realloc                                                      1
  sh_xrealloc                                                     1
  sigon                                                           1
  snprintf                                                        1
[...]
  sigprocmask                                                    16
  __lwp_sigmask                                                  17
  __systemcall                                                   17
  block_all_signals                                              17
```

```
  sigaddset                                                    18
  sigvalid                                                     18
  xstrchr                                                      18
  strchr                                                       23
  strcpy                                                       23
  __sigaction                                                  24
  sigaction                                                    24
  lmutex_lock                                                  26
  lmutex_unlock                                                26
  memset                                                       26
  sh_free                                                      38
  sh_malloc                                                    38
  sh_xfree                                                     38
  sh_xmalloc                                                   38
  strlen                                                       43
  sigemptyset                                                  47
```

5.3. Estadísticas del Kernel

Investigar el uso del comando **kstat.** ¿Qué información nos brinda?

Algunos consumidores y utilitarios construidos por encima de kstat se listan a continuación. Ejecutar dicho comando e investigar que información presentan:

mpstat, vmstat, iostat, netstat, kstat, sar

Para procesos:

prstat, pargs, pflags, pared, pldd, psig, pstack, pmap, pfiles, ptree, ptime, pwdx

Control de procesos:

pgrep, pkill, pstop, prun, pwait, preap

Tracing y debugueo de procesos, threads y kernel:

truss, mdb, dtrace, plockstat, kmdb, lockstat

Información de hardware:

cputrack, cpustat, busstat

5.4. Entrada – Salida

Ver la actividad de base

```
# iostat -cxz 5
                 extended device statistics                      cpu
device    r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b  us sy wt id
  0  0  0 99
sd3       0.1    1.0    4.3   13.7  0.0  0.0   41.3   0   1
nfs56     0.0    0.0    0.7    0.0  0.0  0.0    2.5   0   0
nfs205    0.0    0.0    0.3    0.0  0.0  0.0    6.7   0   0
nfs208    0.0    0.0    0.0    0.0  0.0  0.0    6.0   0   0
                 extended device statistics                      cpu
device    r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b  us sy wt id
  0  1  0 99
sd3       0.0    0.2    0.0    0.2  0.0  0.0    8.6   0   0
```

```
                extended device statistics                      cpu
device    r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b  us sy wt id
  0  1  0 99
sd3       0.0    0.6    0.0    4.8  0.0  0.0   10.2   0   0
                extended device statistics <---- se lanza
                                                                    dd
if=/dev/dsk/c0t0d0s0 of=/dev/null bs=32k <<< Se lanza carga

device    r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b  us sy wt id
  1 10  0 90
sd3     769.7    0.6 6619.7    0.4  0.0  0.8    1.0   0  57
  1 10  0 89
sd3    1170.5    0.6 12147.7    3.4  0.0  1.3    1.1   1  89
  1  6  0 94
sd3     492.4    0.2 12829.5    1.4  0.0  1.3    2.7   0  94
  1 14  0 85
sd3     457.0    0.4 54110.7    2.0  0.0  1.7    3.7   1  99
  5 15  0 80
sd3     383.0    2.2 48777.4   35.6  0.0  2.4    6.1   1 100
  1 14  0 86
sd3     460.1    0.0 50695.5    0.0  0.0  1.6    3.6   1  97
  1 16  0 83
sd3     430.6    9.2 51740.4   23.3  0.0  2.3    5.3   1  99
  1 15  0 84
sd3     402.2    0.0 51229.6    0.0  0.0  1.3    3.3   1  79
```

Se lanza actividad de lectura en disco (comando dd), se ve el uso de CPU que aumenta (el espacio system porque son principalmente system calls), y las lecturas en disco (r/s, lecturas por segundo) lo que se lee en el disco (kr/s, kilobytes por segundos) el porcentaje de utilización de disco (%b, busy), y otros datos muy importantes como el tiempo de servicio (svc_t), las espera (%w) por el dispositivo (el disco), y demás.
Sin la opción 'z' de iostat se verían todos los dispositivos. Al poner la opción 'z', después de mostrar una primera vez todos los dispositivos, muestra solamente los que tienen algún cambio en la actividad.

Para ver los errores o simplemente la existencia de discos:

```
# iostat -E
sd0       Soft Errors: 1 Hard Errors: 2 Transport Errors: 0
Vendor: AMI      Product: Virtual CDROM    Revision: 1.00 Serial No:
Size: 0.00GB <0 bytes>
Media Error: 0 Device Not Ready: 0 No Device: 2 Recoverable: 0
Illegal Request: 1 Predictive Failure Analysis: 0

sd1       Soft Errors: 2 Hard Errors: 0 Transport Errors: 0
Vendor: AMI      Product: Virtual Floppy   Revision: 1.00 Serial No:
Size: 0.00GB <0 bytes>
Media Error: 0 Device Not Ready: 0 No Device: 0 Recoverable: 0
Illegal Request: 2 Predictive Failure Analysis: 0

sd5       Soft Errors: 1 Hard Errors: 0 Transport Errors: 0
Vendor: MATSHITA Product: CD-RW  CW-8124   Revision: DZ13 Serial No:
Size: 0.00GB <0 bytes>
Media Error: 0 Device Not Ready: 0 No Device: 0 Recoverable: 0
Illegal Request: 1 Predictive Failure Analysis: 0
```

De que discos dispongo?

```
# format
Searching for disks...
Inquiry failed on 4
Inquiry failed on 4
done
```

```
AVAILABLE DISK SELECTIONS:
       0. c0t0d0 <DEFAULT cyl 17830 alt 2 hd 255 sec 63>
          /pci@0,0/pci8086,25f8@4/pci108e,286@0/sd@0,0
       1. c0t1d0 <Sun-STK RAID INT-V1.0-409.85GB>
          /pci@0,0/pci8086,25f8@4/pci108e,286@0/sd@1,0
       2. c2t600A0B8000488B3E0000049F499EA4BCd0  <drive type unknown>
          /scsi_vhci/disk@g600a0b8000488b3e0000049f499ea4bc
       3. c2t600A0B8000488F98000004AA499EA44Cd0  <drive type unknown>
          /scsi_vhci/disk@g600a0b8000488f98000004aa499ea44c
Specify disk (enter its number): ^C
```

El comando dtrace es extremadamente poderoso. Mucho de lo relacionado a entrada y salida es opaco al usuario por las varias capas de abstracción que el sistema operativo interpone (drivers, sistema de archivos, etc.). Por ejemplo, quiero saber en mi sistema, en este momento, que se escribe, que se lee y de que tamaño son los bloques de entrada y salida.

Este simple comando dtrace me lo puede decir:

```
#!/usr/sbin/dtrace -s
#pragma D option quiet

dtrace:::BEGIN
{
  printf("%10s %58s %2s %8s \n", "Device", "File", "RW", "Size");
}

io:::start
{
  printf("%10s %58s %2s %8d \n", args[1]->dev_statname,
    args[2]->fi_pathname, args[0]->b_flags & B_READ ? "R" : "W",
    args[0]->b_bcount);
}
```

En la sección BEGIN, se escribe el encabezado del output. Después, por cada entrada y salida (io:::start), se escribe el nombre del dispositivo en el que se opera esta entrada-salida, el path del archivo en el cual se esta operando, si el flag es de lectura, se escribe R, sino W; finalmente el tamaño del bloque que se leyó o escribió.

Este es el output:

```
# ./iotrace.d
    Device                                                File RW     Size
     nfs25                          /home/pvazquez/.sh_history  W     4096
       sd1                 /usr/share/man/ja_JP.UTF-8/man3gen  R     1024
       sd1                 /usr/share/man/ja_JP.UTF-8/man3lib  R     1024
       sd1                /usr/share/man/ja_JP.UTF-8/man3tsol  R     2048
       sd1                                            <none>  R     8192
       sd1                   /usr/share/man/ja_JP.UTF-8/man7d  R     1024
       sd1                   /usr/share/man/ja_JP.UTF-8/man7i  R     1024
       sd1                   /usr/share/man/ja_JP.UTF-8/man7m  R     1024
       sd1                   /usr/share/man/ja_JP.UTF-8/man9f  R     1024
       sd1                                            <none>  R     8192
       sd1                      /usr/share/man/man3fontconfig  R     7168
       sd1                                            <none>  R     8192
       sd1                                            <none>  R     8192
       sd1                                 /usr/share/man/zh  R     1024
       sd1                            /usr/share/man/zh/man1  R     1024
```

Fíjense que la primera línea es de escritura, y el resto de lectura. La primera línea indica que la operación se hizo vía NFS (`nfs25`) o sea que la operación se hizo en un dispositivo remoto, vía la red. El resto fue en dispositivos locales; mayormente lectura de man pages.

Recién vimos un ejemplo donde podemos saber donde se opera y el tamaño de la operación. Pero cuantas veces se da ese tamaño? (no es lo mismo escribir una vez un bloque de 128K que 1000 veces 1 bloque de 64K). Cada aplicación tiene su perfil operativo. Algunas escribirán más, otras serán más bien de lectura. Unas operaran con bloques grandes, otras con todo tipo de tamaño. La pregunta es entonces: cuál es el perfil operativo de los procesos que están corriendo? Para eso, lo que queremos saber es cual es la distribución estadística de los tamaños de bloques que intervinieron en las entrada-salidas, y a que proceso pertenece cada distribución.

Para ver la distribución por tamaño de bloques escrito:

```
#!/usr/sbin/dtrace -s
#pragma D option quiet

dtrace:::BEGIN
{
  printf("Tracing... Hit Ctrl-C to end.\n");
}

io:::start
{
  @size[pid, curpsinfo->pr_psargs] = quantize(args[0]->b_bcount);
}

dtrace:::END
{
  printf("%8s  %s\n", "PPID", "CMD");
  printa("%8d  %s\n%@d\n", @size);
}
```

Este es el output:

```
# ./bytes.d
Tracing... Hit Ctrl-C to end.
^C
    PPID  CMD
   22043  /usr/sbin/dtrace -s ./bytes.d

           value  ------------- Distribution ------------- count
            2048 |                                         0
            4096 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 4
            8192 |                                         0

       3  fsflush

           value  ------------- Distribution ------------- count
            2048 |                                         0
            4096 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1758
            8192 |                                         0

     581  /usr/lib/fm/fmd/fmd

           value  ------------- Distribution ------------- count
             256 |                                         0
             512 |@@@@@@@@@@@@@@@@@@@@@@@@@@               37545
            1024 |                                         87
            2048 |                                         189
```

```
            4096 |@@@@@@@@@@                               15712
            8192 |@@@@@@                                   9343
           16384 |                                         0

       0  sched

           value  ------------- Distribution -------------  count
             256 |                                         0
             512 |                                         1
            1024 |                                         1
            2048 |@@@@@@@@@@                               75628
            4096 |@@@@@@@@@@@@@@@@@@@@@@@@@@               179360
            8192 |@@@@@@                                   43183
           16384 |                                         1896
           32768 |                                         0
```

Este output muestra varios procesos con su distribución de operaciones asociada. Por ejemplo el proceso fmd, opera en su mayoría con bloques de 512 bytes, y casi la mitad de esas veces con bloques de 4K. Mientras que el perfil de uso del proceso sched opera en su mayoría con bloques de 4K, y el resto de 2 y 8K.

Como dijimos, dtrace permite ver cosas que son extremadamente difíciles de cuantificar. Los famosos 'seek' del disco. Cuando se lee o escribe un bloque se desea que el próximo bloque que se vaya a operar, este cerca del que estoy haciendo. En otras palabras, que la búsqueda del próximo bloque sea lo más corta posible. Nunca antes se vio. Acá lo tienen, midiendo las distancias de seek (de búsqueda) entre operación y operación y su distribución:

```
#!/usr/sbin/dtrace -s
#pragma D option quiet

self int last[dev_t];

dtrace:::BEGIN
{
  printf("Tracing... Hit Ctrl-C to end.\n");
}

io:genunix::start
/self->last[args[0]->b_edev] != 0/
{
  this->last = self->last[args[0]->b_edev];
  this->dist = (int)(args[0]->b_blkno - this->last) > 0 ?
       args[0]->b_blkno - this->last : this->last - args[0]->b_blkno;
  @size[args[1]->dev_statname] = quantize(this->dist);

}

io:genunix::start
{
  self->last[args[0]->b_edev] = args[0]->b_blkno +
        args[0]->b_bcount / 512;
}
```

Acá está el output:

```
# ./seek.d
Tracing... Hit Ctrl-C to end.
^C
```

```
sd5
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@                                     17780
               1 |@@                                       8095
               2 |@                                        2534
               4 |@                                        3696
               8 |@@@                                      12241
              16 |@@@@@@@                                  28649
              32 |@@@@@@@@@@                               42553
              64 |@@@@@@                                   25988
             128 |@@@@                                     16394
             256 |@@                                       6916
             512 |@                                        3811
            1024 |                                         1278
            2048 |                                         91
            4096 |                                         1
            8192 |                                         0

sd6
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@                                     17607
               1 |@@                                       8147
               2 |@                                        2580
               4 |@                                        3743
               8 |@@@                                      12271
              16 |@@@@@@@                                  28955
              32 |@@@@@@@@@@                               42243
              64 |@@@@@@                                   25814
             128 |@@@@                                     16538
             256 |@@                                       6850
             512 |@                                        3766
            1024 |                                         1283
            2048 |                                         110
            4096 |                                         0

sd1
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@                                        2015
               1 |@@@@@@@@                                 17113
               2 |                                         773
               4 |                                         2
               8 |                                         19
              16 |@                                        2788
              32 |                                         10
              64 |                                         16
             128 |                                         16
             256 |                                         21
             512 |                                         24
            1024 |                                         53
            2048 |                                         95
            4096 |                                         151
            8192 |                                         229
           16384 |                                         220
           32768 |                                         124
           65536 |                                         456
          131072 |                                         192
          262144 |                                         44
          524288 |                                         51
         1048576 |                                         7
```

```
 2097152 |                                        0
 4194304 |                                        1
 8388608 |                                        150
16777216 |                                        726
33554432 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@       55961
67108864 |                                        0
```

Como es lógico, las distribuciones se presentan por dispositivo de la máquina (sd1,sd5, sd6,….). Por ejemplo vemos que en el dispositivo sd1, la gran mayoría de las búsquedas del próximo bloque tardaron entre 33554432 y 67108864 unidades de tiempo. Mientras que en el resto de los dispositivos, encontramos una típica campana de Gauss (o sea una distribución normal) que va de más o menos 4 a 512 con su pico en 32. En síntesis el uso del dispositivo sd1 es diferente del resto de los dispositivos. En promedio todos tardan entre 32 y 64 unidades de tiempo en buscar el próximo bloque, mientras que en sd1 se tarda entre 33554432 y 67108864. Esto podría indicar que en el dispositivo sd1, un reordenamiento de los bloques mejoraría la performance o que el driver que se usa no es óptimo, o que las aplicaciones deberían cambiar la forma en que acceden a los archivos.