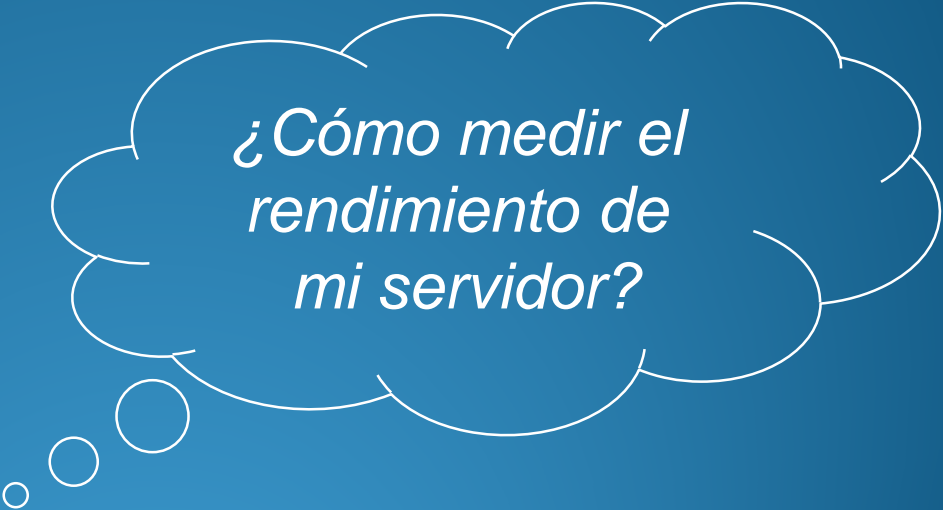


Tema 3. Monitorización de servicios y programas



*¿Cómo medir el
rendimiento de
mi servidor?*

Analistas, administradores y diseñadores



Objetivos del tema

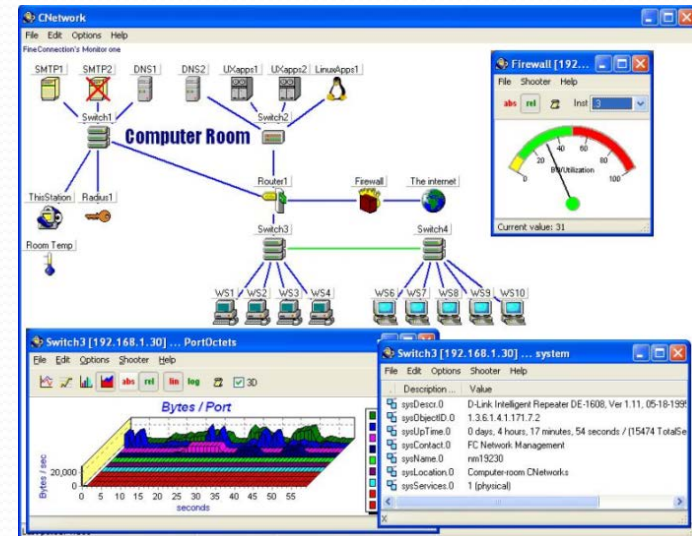
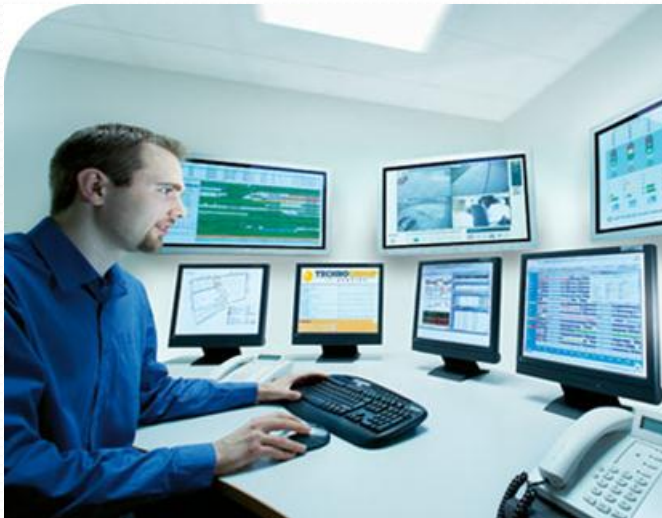
- Entender el concepto de monitor de actividad de un servidor y sus diferentes utilidades e implementaciones.
- Conocer las características fundamentales de un monitor a nivel de sistema operativo y a nivel de aplicación concreta (profilers).
- Comprender el papel que desempeñan los monitores para evaluar el rendimiento de un servidor ante una carga real.
- Saber interpretar adecuadamente la información que aporta un monitor.

Bibliografía

- *Evaluación y modelado del rendimiento de los sistemas informáticos*. Xavier Molero, C. Juiz, M. Rodeño. Pearson Educación, 2004.
 - Capítulo 2
- *Measuring computer performance: a practitioner's guide*. D. J. Lilja, Cambridge University Press, 2000.
 - Capítulos 4 y 6
- *The art of computer system performance analysis*. R. Jain. John Wiley & Sons, 1991.
 - Capítulos 7 y 8
- *System performance tuning*. G.D. Musumeci, M. Loukides. O'Reilly, 2002.
 - Capítulo 2
- *Linux performance and tuning guidelines*. E.Ciliendo, T.Kunimasa. IBM Redpaper, 2007.
 - Capítulos 1 y 2
- *Linux performance tools*. B. Gregg. 2015. <https://conferences.oreilly.com/velocity/devops-web-performance-2015/public/schedule/detail/42513>.
- *Linux man pages*. <http://www.linuxmanpages.com/>.

Contenido

- Concepto de monitor de actividad.
- Monitorización a nivel de sistema.
- Monitorización a nivel de aplicación.



3.1. Concepto de Monitor de Actividad

La carga y la actividad de un sistema

- **Carga** (*workload*): conjunto de tareas que ha de realizar un sistema. (= Todo aquello que demande recursos del sistema.)
- **Actividad** de un sistema: conjunto de operaciones que se realizan en el sistema como consecuencia de la carga que soporta.
- Un sistema informático no es “bueno” ni “malo” *per se*, sino que se adapta mejor o peor a un tipo determinado de carga.
- Algunas variables que reflejan la actividad de un sistema:
 - Procesadores: Utilización, temp., f_{CLK} , nº de procesos, nº de interrupciones, cambios de contexto, etc.
 - Memoria: nº de accesos, memoria utilizada, fallos de caché, fallos de página, uso de memoria de intercambio, latencias, anchos de banda, voltajes, etc.
 - Discos: lecturas/escrituras por unidad de tiempo, longitud de las colas de espera, tiempo de espera medio por acceso, etc.
 - Red: paquetes recibidos/enviados, colisiones por segundo, sockets abiertos, paquetes perdidos, etc.
 - Sistema global: nº de usuarios, nº de peticiones, etc.



Definición de monitor de actividad

- Herramienta diseñada para medir la actividad de un sistema informático y facilitar su análisis.



- Acciones típicas de un monitor:
 - Medir la actividad.
 - Procesar y almacenar la información recopilada.
 - Mostrar los resultados.



Utilidad de los monitores de actividad

- Administrador/Ingeniero
 - Conocer la utilización de los recursos (detección de cuellos de botella) para saber:
 - Qué hardware hay que reconfigurar / sustituir/ añadir.
 - Cómo ajustar los parámetros del sistema (sintonización).
 - Predecir cargas futuras (capacity planning).
 - Tarificar a los clientes.
 - Obtener un modelo un componente o de todo el sistema para poder deducir qué pasaría si...
- Programador
 - Conocer las partes críticas de una aplicación de cara a su optimización.
- Sistema Operativo
 - Adaptarse dinámicamente a la carga.



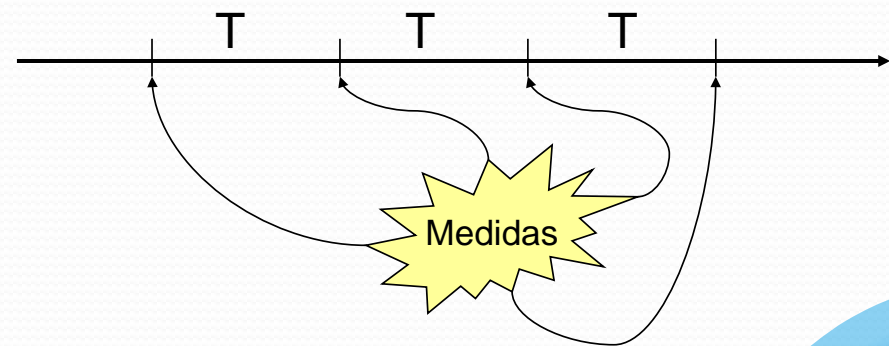
Tipos de monitores: ¿cuándo se mide?

Cada vez que ocurre un evento (*monitor por eventos*)

- Evento: Cambio en el estado del sistema.
 - Volumen de información recogida: Depende de la frecuencia de los eventos.
 - Información exacta.
- Ejemplos de eventos:
 - Inicio/fin de la ejecución de un programa.
 - Fallo en memoria cache.
 - Interrupción de un dispositivo periférico.
 - Abrir/cerrar un fichero, etc.

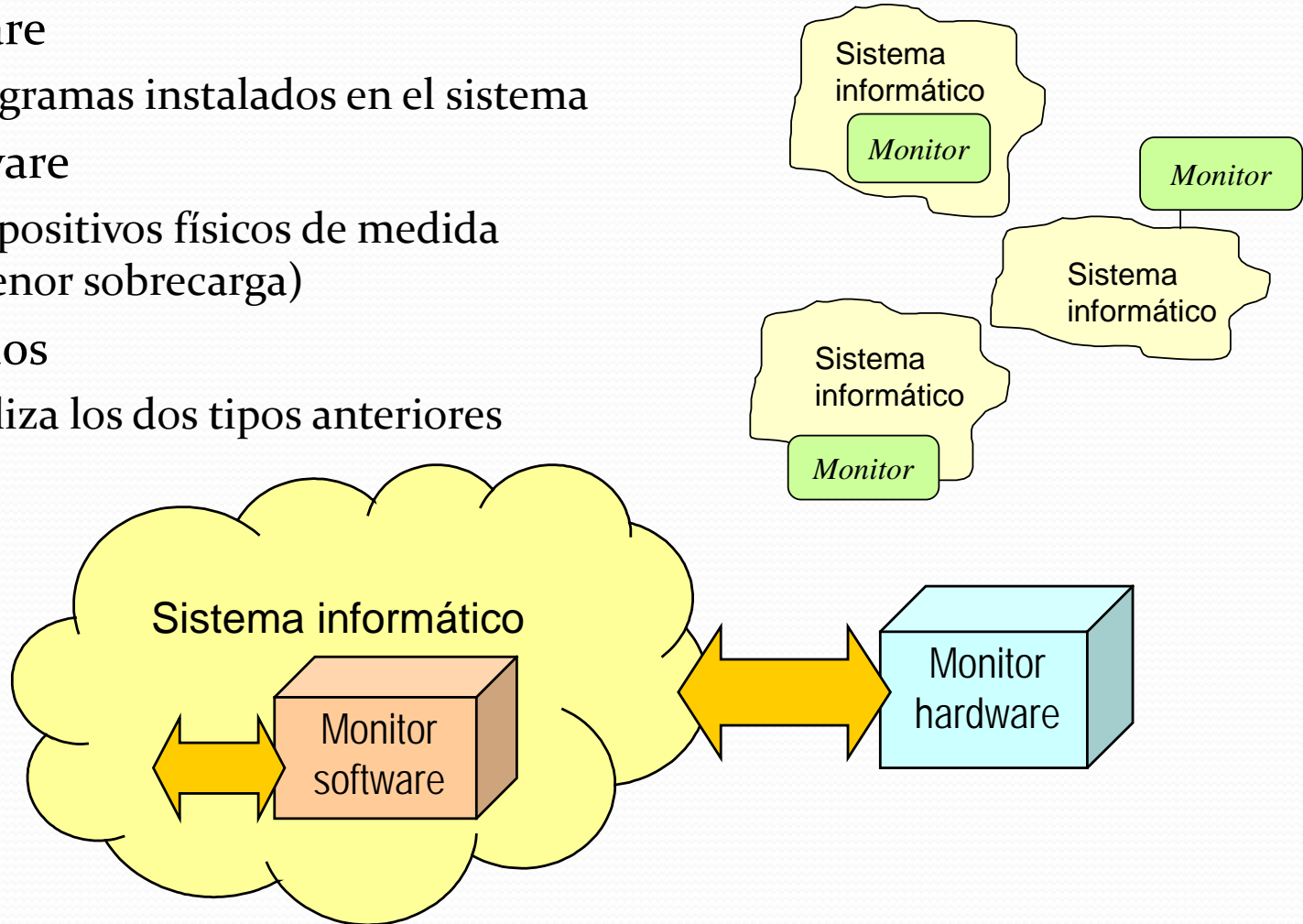
A intervalos regulares de tiempo (*monitor por muestreo*)

- Volumen de información recogida: Depende del periodo de muestreo (T).
- Información estadística.



Tipos de monitores: ¿cómo se mide?

- Software
 - Programas instalados en el sistema
- Hardware
 - Dispositivos físicos de medida (menor sobrecarga)
- Híbridos
 - Utiliza los dos tipos anteriores



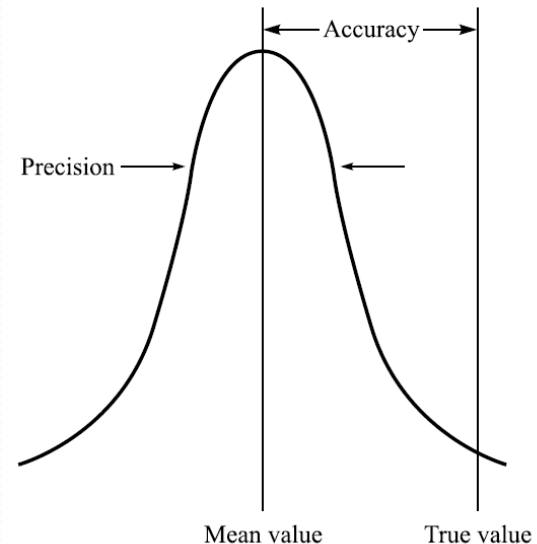
Tipos de monitores: ¿existe interacción con el analista/administrador?

- No existe. La consulta sobre los resultados se realiza aparte mediante otra herramienta independiente al proceso de monitorización: monitores tipo batch, por lotes o en segundo plano (*batch monitors*).
- Sí existe. Durante el propio proceso de monitorización se pueden consultar los valores monitorizados y/o interactuar con ellos realizando representaciones gráficas diversas, modificando parámetros del propio monitor, etc.: monitores en primer plano o interactivos (*on-line monitors*).



Atributos que caracterizan a un sensor/monitor

- **Exactitud** de la medida (*Accuracy, offset*): ¿Cómo se aleja el valor medido del valor real que se quiere medir?
- **Precisión** (*Precision*): ¿Cuál es la dispersión de las medidas?
- **Resolución** del sensor: ¿Cuánto tiene que cambiar el valor a medir para detectar un cambio?
- **Tasa Máxima de Entrada** (*Max Input Rate*): ¿Cuál es la frecuencia máxima de ocurrencia de los eventos que el monitor puede observar? (monitores por eventos)
- **Periodo de Muestreo** (*Sampling Time*): ¿Cada cuánto tiempo tomamos la medida? (monitores por muestreo)
- **Anchura de Entrada** (*Input Width*): ¿Cuánta información (nº de bytes de datos) se almacena por cada medida que toma el monitor?



Más atributos: sobrecarga (overhead)

- **Sobrecarga (Overhead):** ¿Qué recursos le “roba” el monitor al sistema?

➡ *El instrumento de medida puede perturbar el funcionamiento del sistema.*

$$\text{Sobrecarga}_{\text{Recurso}}(\%) = \frac{\text{Uso del recurso por parte del monitor}}{\text{Capacidad total del recurso}} \times 100$$

- Ejemplo: Sobrecarga de CPU de un monitor software por muestreo. La ejecución de las instrucciones del monitor se lleva a cabo utilizando recursos del sistema monitorizado. Supongamos que el monitor se activa cada 5 s y cada activación del mismo usa el procesador durante 6 ms.

$$\text{Sobrecarga}_{\text{CPU}}(\%) = \frac{6 \times 10^{-3} \text{ s}}{5 \text{ s}} \times 100 = 0,12\%$$

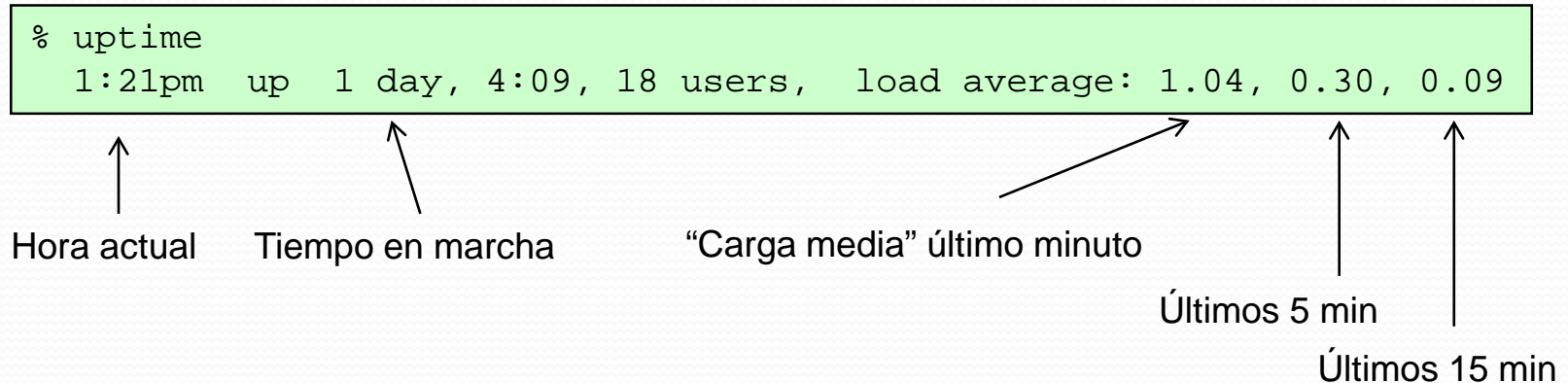
3.2. Monitorización a nivel de sistema

El directorio `/proc` (Unix)

- Es una carpeta en RAM utilizada por el núcleo de Unix para facilitar el acceso del usuario a las estructuras de datos del S.O.
- A través de `/proc` podemos:
 - Acceder a información global sobre el S.O. : `loadavg`, `uptime`, `cpuinfo`, `meminfo`, `mounts`, `net`, `kmsg`, `cmdline`, `slabinfo`, `filesystems`, `diskstats`, `devices`, `interrupts`, `stat`, `swap`, `version`, `vmstat` ...
 - Acceder a la información de cada uno de los procesos del sistema (`/proc/[pid]`): `stat`, `status`, `statm`, `mem`, `smaps`, `cmdline`, `cwd`, `environ`, `exe`, `fd`, `task`...
 - Acceder y, a veces, modificar algunos parámetros del kernel del S.O. (`/proc/sys`): `dentry_state`, `dir-notify-enable`, `dquot-max`, `dquot-nr`, `file-max`, `file-nr`, `inode-max`, `inode-nr`, `lease-break-time`, `mqueue`, `super-max`, `super-nr`, `acct`, `domainname`, `hostname`, `panic`, `pid_max`, `version`, `net`, `vm`...
- La mayoría de los monitores de Unix usan como fuente de información este directorio.

uptime

- Tiempo que lleva el sistema en marcha y la “carga media” que soporta



[man uptime \(http://man7.org/linux/man-pages/man1/uptime.1.html\)](http://man7.org/linux/man-pages/man1/uptime.1.html)

NAME

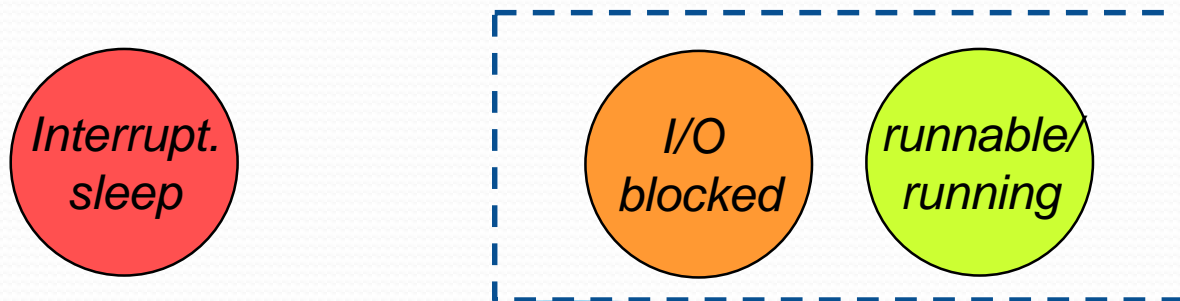
uptime - Tell how long the system has been running.

DESCRIPTION

- **uptime** gives a one line display of the following information. The current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes. This is the same information contained in the header line displayed by `w`.

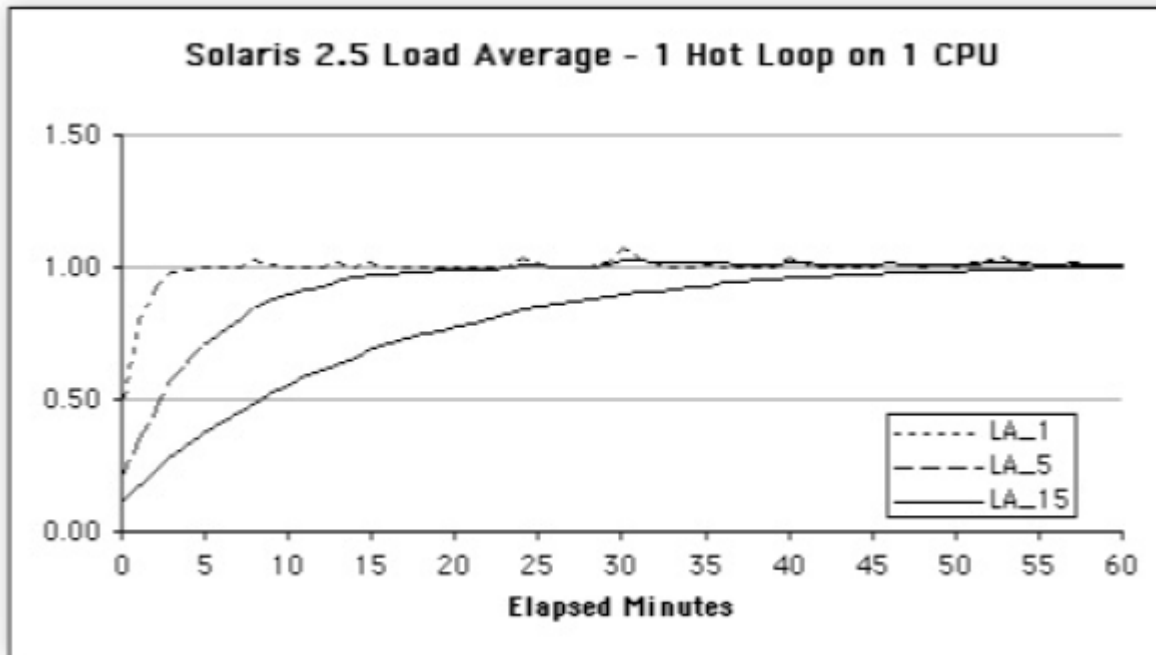
“Carga” del sistema Unix

- Estados básicos de un proceso:
 - En ejecución (*running*) o en la cola de ejecución (*runnable*).
 - Bloqueado esperando a que se complete una operación de E/S para continuar (*uninterruptible sleep* = *I/O blocked*).
 - Durmiendo esperando a un evento del usuario o similar (p.ej. una pulsación de tecla) (*interruptible sleep*).
- La cola de procesos del núcleo (*run queue*) está formada por aquellos que pueden ejecutarse (*runnable* + *running*).
- “Carga del sistema” (*system load*): número de procesos en modo *running*, *runnable* o *I/O blocked*.



¿Cómo mide la carga media el S.O.?

- Experimento: Ejecutamos 1 único proceso (bucle infinito). Llamamos a uptime cada cierto tiempo y representamos los resultados.



Según timer.c
(kernel de Linux):
$$LA(t) = c \cdot \text{load}(t) + (1-c) \cdot LA(t-5)$$

LA = Load Average
Se actualiza cada 5 segundos su valor.

- c es una constante. A mayor valor, más influencia tiene la carga actual en el valor medio ($c_1 > c_5 > c_{15}$). Si $c = c_1$ calculamos $LA_1(t)$, etc.
- $\text{load}(t)$ es la “carga del sistema” en el instante t .

ps (*process status*)

- Información sobre el estado actual de los procesos del sistema.

```
$ ps aur
USER      PID %CPU %MEM  VSZ   RSS TTY      STAT START  TIME COMMAND
miguel    29951 55.9  0.1 1448   384 pts/0    R      09:16  0:11 tetris
carlos    29968 50.6  0.1 1448   384 pts/0    R      09:32  0:05 tetris
javier    30023  0.0  0.5 2464  1492 pts/0    R      09:27  0:00 ps aur
```

- Se pueden seleccionar multitud de campos diferentes que mostrar sobre cada proceso y diferentes formas de ordenar la lista resultante.

Procesos a mostrar:

-A, -e: show all processes; T: all processes on this terminal
U: processes for specified users...

Orden de presentación:

-C: sort by command name, -p sort by process ID...

Campos que mostrar: simple name of executable, full command line, process group ID, cumulative user time, user time, s, number of minor/major page faults, session ID, process ID, parent process ID, resident set size, memory size in kilobytes, amount of shared pages, time process was started, user ID number, user name, total VM size in bytes, kernel scheduling priority, etc.

Información aportada por ps

- USER
 - Usuario que lanzó el proceso
- %CPU, %MEM
 - Porcentaje de procesador y memoria física usada
- SIZE (o VSIZE)
 - Memoria (KiB) virtual total asignada al proceso
- RSS (*resident set size*)
 - Memoria (KiB) física ocupada por el proceso
- STAT
 - R (*running or runnable*), D (*I/O blocked*), S (*interruptible sleep*), T (*stopped*), Z (*zombie: terminated but not died*)
 - N (*lower priority = running niced*), < (*higher priority = not nice*)
 - s (*session leader*), + (*in the foreground process group*), W (*swapped*)

top

- Muestra cada T segundos: carga media, procesos, consumo de memoria...
- Normalmente se ejecuta en modo interactivo (se puede cambiar T, las columnas seleccionadas, la forma de ordenar las filas, etc.)

```
8:48am up 70 days, 21:36, 1 user, load average: 0.28, 0.06, 0.02
Tareas: 47 total, 2 running, 45 sleeping, 0 stopped, 0 zombie
%CPU(s): 99.6% user, 0.3% system, 0.0% nice, 0.0% idle
Mem: 256464K av, 234008K used, 22456K free, 0K shrd, 13784K buff
Swap: 136512K av, 4356K used, 132156K free 5240K cached
```

PID	USER	PR	NI	VIRT	RES	SHR	STAT	%CPU	%MEM	TIME	COMMAND
9826	carlos	0	0	388	388	308	R	99.6	0.1	0:22	simulador
9831	miguel	19	0	976	976	776	R	0.3	0.3	0:00	top
1	root	20	0	76	64	44	S	0.0	0.0	0:03	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00	keventd
4	root	20	19	0	0	0	SN	0.0	0.0	0:00	ksoftiq
5	root	20	0	0	0	0	S	0.0	0.0	0:13	kswapd
6	root	2	0	0	0	0	S	0.0	0.0	0:00	bdfldsh
7	root	20	0	0	0	0	S	0.0	0.0	0:10	kdated
8	root	20	0	0	0	0	S	0.0	0.0	0:01	kinoded
11	root	0	-20	0	0	0	S<	0.0	0.0	0:00	recoved

vmstat (*virtual memory statistics*)

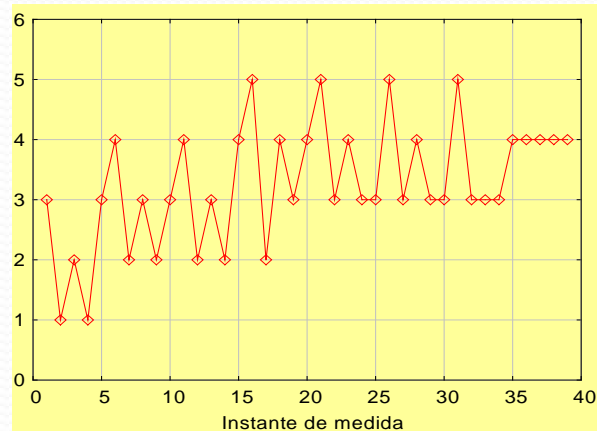
- *Paging* (paginación), *swapping*, interrupciones, cpu
 - La primera línea no sirve para nada (info desde el inicio del sistema)

% vmstat 1 6																	
procs			memory				swap		io		system		cpu				
r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st	
0	0	868	8964	60140	342748	0	0	23	7	222	199	1	4	80	15	0	
0	0	868	8964	60140	342748	0	0	0	14	283	278	0	7	80	23	0	
0	0	868	8964	60140	342748	0	0	0	0	218	212	6	2	93	0	0	
0	0	868	8964	60140	342748	0	0	0	0	175	166	3	3	94	0	0	
0	0	868	8964	60140	342752	0	0	0	2	182	196	0	7	88	5	0	
0	0	868	8968	60140	342748	0	0	0	18	168	175	3	8	69	20	0	

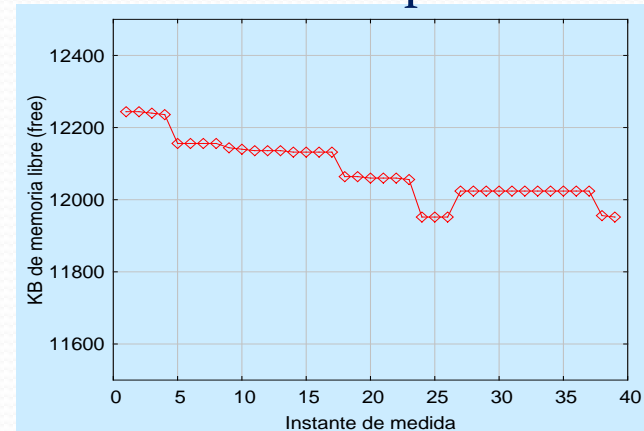
- Procesos: *r* (*runnable*), *b* (*I/O blocked*)
- Bloques por segundo transmitidos: *bi* (*blocks in*), (*blocks out*)
- KB/s entre memoria y disco: *si* (*swapped in*), *so* (*swapped out*)
- *in* (*interrupts per second*), *cs* (*context switches per second*)
- Con otros argumentos, puede dar información sobre acceso a discos (en concreto la partición de swap) y otras estadísticas de memoria.

Representación gráfica de los resultados

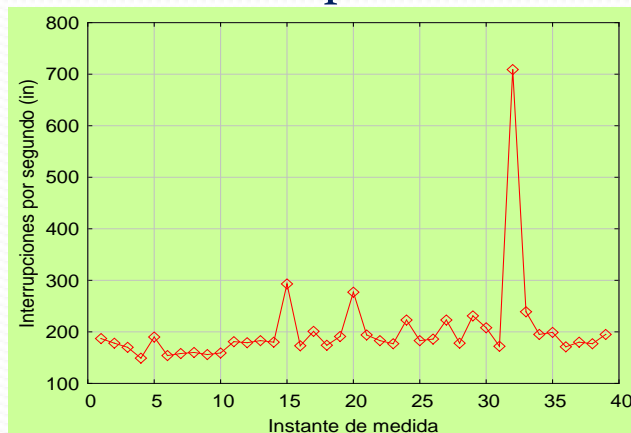
Procesos activos



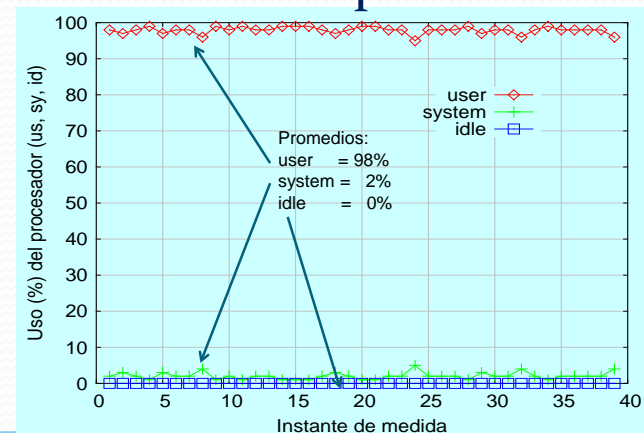
Memoria disponible



Interrupciones



Utilización procesador



El paquete de monitorización: SysStat



The screenshot shows the SysStat Utilities Home Page in a web browser. The page has a dark blue background with a header section featuring a penguin, a computer tower, and a monitor displaying 'LINUX'. The text 'Welcome to the SYSSTAT Utilities Home Page!' is prominently displayed. Below the header is a navigation bar with links: News, Features, Download, GitHub, Documentation, Tutorial, FAQ, and Contact. A 'Useful links' section on the left lists various Linux-related resources. The main content area contains a paragraph about the SysStat utilities, their features, and their licensing. A 'News' section follows, dated Monday August 14th, 2017, listing several releases of SysStat and details about version 11.6.0. At the bottom left, there is a 'Donate' button, a W3C HTML 4.01 logo, and a visitor counter showing 667220.

☆ | 🏠 | 🔄 | sebastien.godard.pagesperso-orange.fr | 150% | 🔍 sysstat

Welcome to the
SYSSTAT
Utilities Home Page!

[News](#) | [Features](#) | [Download](#) | [GitHub](#) | [Documentation](#) | [Tutorial](#) | [FAQ](#) | [Contact](#)

Useful links:
[Freecode](#)
[IceWalkers](#)
[Qwant](#)

[The Linux Foundation](#)
[Linux Kernel Archives](#)
[Linux Kernel Mailing List](#)
[The Linux Documentation Project](#)

This page has news, information, documentation and links software for the **sysstat** utilities that I created for Linux. The sysstat utilities are a collection of **performance monitoring tools** for Linux. These include **sar, sadf, mpstat, iostat, tapestat, pidstat, cifsioat and sa tools**. Go to the [Features](#) page to display a list of sysstat's features, or see the [Documentation](#) page to learn some more about them.
Sysstat is [Open Source](#) / [Free Software](#), and is freely available under the [GNU General Public License, version 2](#).

News

Monday August 14th, 2017

Sysstat 11.6.0 released (stable version).
Sysstat 11.4.6 released (stable version).
Sysstat 11.2.12 released (stable version).
Version 11.6.0 (the first of a new stable series) mainly focuses on the pidstat command, with the addition of two new options:
Option -e can be used to pass a program to execute and make pidstat monitor it, whereas option -H tells pidstat to display timestamps in seconds since the Epoch.
Also these versions revert a fix (added in previous sysstat version to fix a possible sadc crash on ARM architecture) that could cause segmentation faults in some cases.

Donate

W3C HTML 4.01

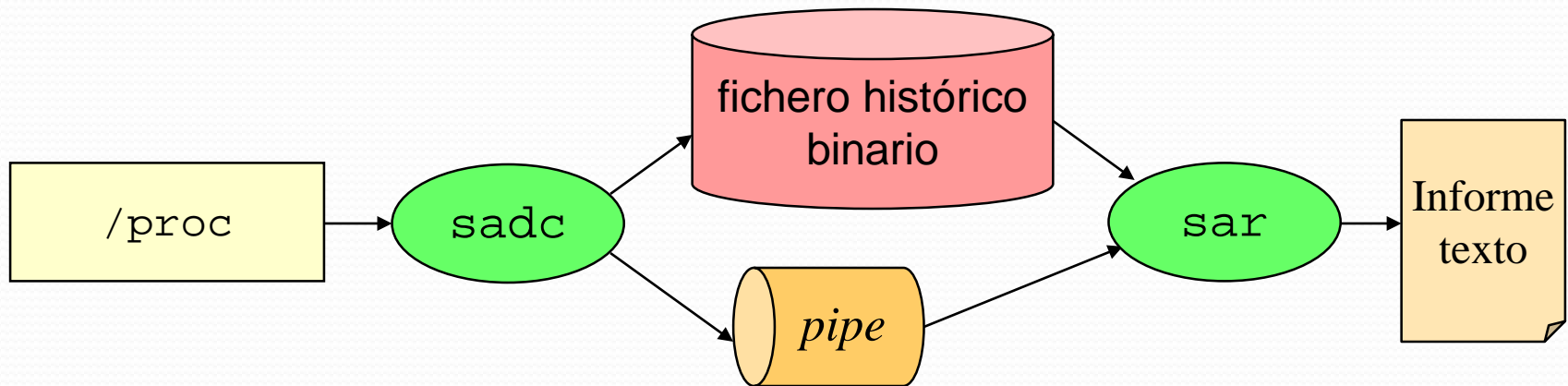
You are visitor #
667220

El monitor sar

- sar (*system activity reporter*)
 - Es un monitor de actividad utilizado por los administradores de sistemas Unix para, entre otras cosas, la detección de cuellos de botella (*bottlenecks*).
 - Información sobre todo el sistema.
 - Actual: qué está pasando el día de hoy, o ahora mismo, en el sistema.
 - Histórica: qué ha pasado en el sistema en otros días pasados.
 - Ficheros históricos en `/var/log/sysstat/saDD`, donde los dígitos DD indican el día del mes.
 - Hace uso de contadores estadísticos del núcleo del sistema operativo ubicados en el directorio `/proc`.
- Disponibilidad en Internet y ya incluido en muchas distribuciones de Linux.
 - <http://sebastien.godard.pagesperso-orange.fr/>

Funcionamiento del monitor sar

- Se basa en dos órdenes complementarias:
 - `sadc` (*system-accounting data collector*)
 - Recoge los datos estadísticos (lectura de contadores) y construye un registro en formato binario (*back-end*)
 - `sar`
 - Lee los datos binarios que recoge `sadc` y los traduce a un formato legible por nosotros en formato texto (*front-end*)



Parámetros de sar

- Gran cantidad de parámetros (puede funcionar en modo batch o en modo interactivo)

Modo interactivo: [tiempo_muestreo, [nº muestras]]

Modo no interactivo:

-f Fichero de donde extraer la información, **por defecto: hoy**
-s Hora de comienzo de la monitorización
-e Hora de fin de la monitorización

-u Utilización del procesador (**opción por defecto**)
-P Mostrar estadísticas por cada procesador (-P ALL: todos)
-I Estadísticas sobre interrupciones
-w Cambios de contexto
-q Tamaño de la cola y carga media del sistema
-b Estadísticas de transferencias de E/S
-d Transferencias para cada disco
-n Conexión de red
-r Utilización de memoria
-R Estadísticas sobre la memoria
-A Toda la información disponible
...

Ejemplo de salidas del monitor sar

- Utilización del procesador (que puede ser multi-núcleo):

```
$ sar (=sar -u)
00:00:00      CPU      %user      %nice      %system      %iowait      %steal      %idle
00:05:00     all       0.09       0.00       0.08         0.00         0.00       99.83
00:10:00     all       0.01       0.00       0.01         0.00         0.00       99.98
...
11:15:00     all       0.02       0.00       0.02         0.00         0.00       99.96
11:20:00     all       0.44       0.00       0.20         0.00         0.00       99.36
11:25:00     all       0.05       0.00       0.02         0.00         0.00       99.92
```

- Actividad del sistema de entrada/salida

```
$ sar -b
00:00:00          tps          rtps          wtps        bread/s        bwrtn/s
00:05:00          0.74          0.39          0.35          7.96           3.27
00:10:01          0.09          0.00          0.09          0.00           0.91
00:15:00          0.15          0.00          0.14          0.03           1.36
00:20:00        65.12        59.96          5.16        631.62        162.64
```


Otros ejemplos de ejecución de sar

- Ejecución interactiva (tiempo de muestro, nº muestras)

```
$ sar 1 3
19:20:39      CPU      %user      %nice      %system      %iowait      %steal      %idle
19:20:40     all       11.88       0.00       2.97       0.00       0.00      85.15
19:20:41     all       21.00       0.00       2.00       0.00       0.00      77.00
19:20:42     all       20.41       0.00       1.02       0.00       0.00      78.57
```

- Información recogida sobre el día de hoy

- sar (= sar -u)
- sar -d -s 10:00:00 -e 12:00:00
- sar -A
- sar -b

- Información recogida en otro día anterior

- sar -f /var/log/sysstat/sa02
- sar -P ALL -f /var/log/sysstat/sa06
- sar -d -s 10:00:00 -e 12:00:00 -f /var/log/sa/sa04

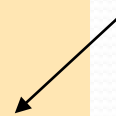
Los datos sobre la actividad

- Se utiliza un fichero histórico de datos por cada día.
- Se programa la ejecución de `sadc` un número de veces al día con la utilidad “cron” del sistema Unix.
 - Por ejemplo, una vez cada 5 minutos comenzando a las 0:00 de cada día.
- Cada ejecución de `sadc` añade un registro binario con los datos recogidos al fichero histórico del día.

```
%ls /var/log/sysstat
```

```
-rw-r--r-- 1 root root 3049952 Sep 30 23:55 sa30
-rw-r--r-- 1 root root 3049952 Oct 1 23:55 sa01
-rw-r--r-- 1 root root 3049952 Oct 2 23:55 sa02
-rw-r--r-- 1 root root 3049952 Oct 3 23:55 sa03
-rw-r--r-- 1 root root 3049952 Oct 4 23:55 sa04
-rw-r--r-- 1 root root 3049952 Oct 5 23:55 sa05
-rw-r--r-- 1 root root 3049952 Oct 6 23:55 sa06
-rw-r--r-- 1 root root 3049952 Oct 7 23:55 sa07
-rw-r--r-- 1 root root 2372320 Oct 8 18:45 sa08
```

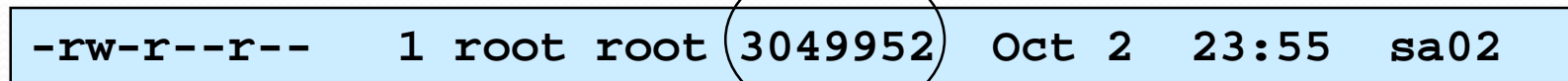
Día actual



Análisis de un fichero histórico

- Ejemplo

- El fichero histórico de un día ocupa 3.049.952 bytes (unos 3 MB)
- La orden `sadc` se ejecuta cada 5 minutos
 - Cada hora se recogen 12 muestras
 - Al día se recogen $24 \times 12 = 288$ muestras
- Anchura de entrada del monitor:
Cada registro ocupa de media 10,3 KiB



```
-rw-r--r--  1 root root 3049952 Oct 2 23:55 sa02
```

Fichero `sa02`
(día 2 de octubre)

Otras herramientas de Sysstat

- `mpstat` (*processors related statistics*). Nos muestra estadísticas concretas por cada núcleo (core). Podemos ver si la carga está bien distribuida entre los diferentes núcleos.

```
$ mpstat -P ALL 1
[...]
```

08:06:43	PM	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%idle
08:06:44	PM	all	53.45	0.00	3.77	0.00	0.00	0.39	0.13	0.00	42.26
08:06:44	PM	0	49.49	0.00	3.03	0.00	0.00	1.01	1.01	0.00	45.45
08:06:44	PM	1	51.61	0.00	4.30	0.00	0.00	2.15	0.00	0.00	41.94
08:06:44	PM	2	58.16	0.00	7.14	0.00	0.00	0.00	1.02	0.00	33.67
08:06:44	PM	3	54.55	0.00	5.05	0.00	0.00	0.00	0.00	0.00	40.40
08:06:44	PM	4	47.42	0.00	3.09	0.00	0.00	0.00	0.00	0.00	49.48
08:06:44	PM	5	65.66	0.00	3.03	0.00	0.00	0.00	0.00	0.00	31.31
08:06:44	PM	6	50.00	0.00	2.08	0.00	0.00	0.00	0.00	0.00	47.92

```
[...]
```

Otras herramientas de Sysstat (cont.)

- `iostat` (*input/output statistics*). Muestra estadísticas de los dispositivos de E/S y particiones de disco existentes en el sistema.

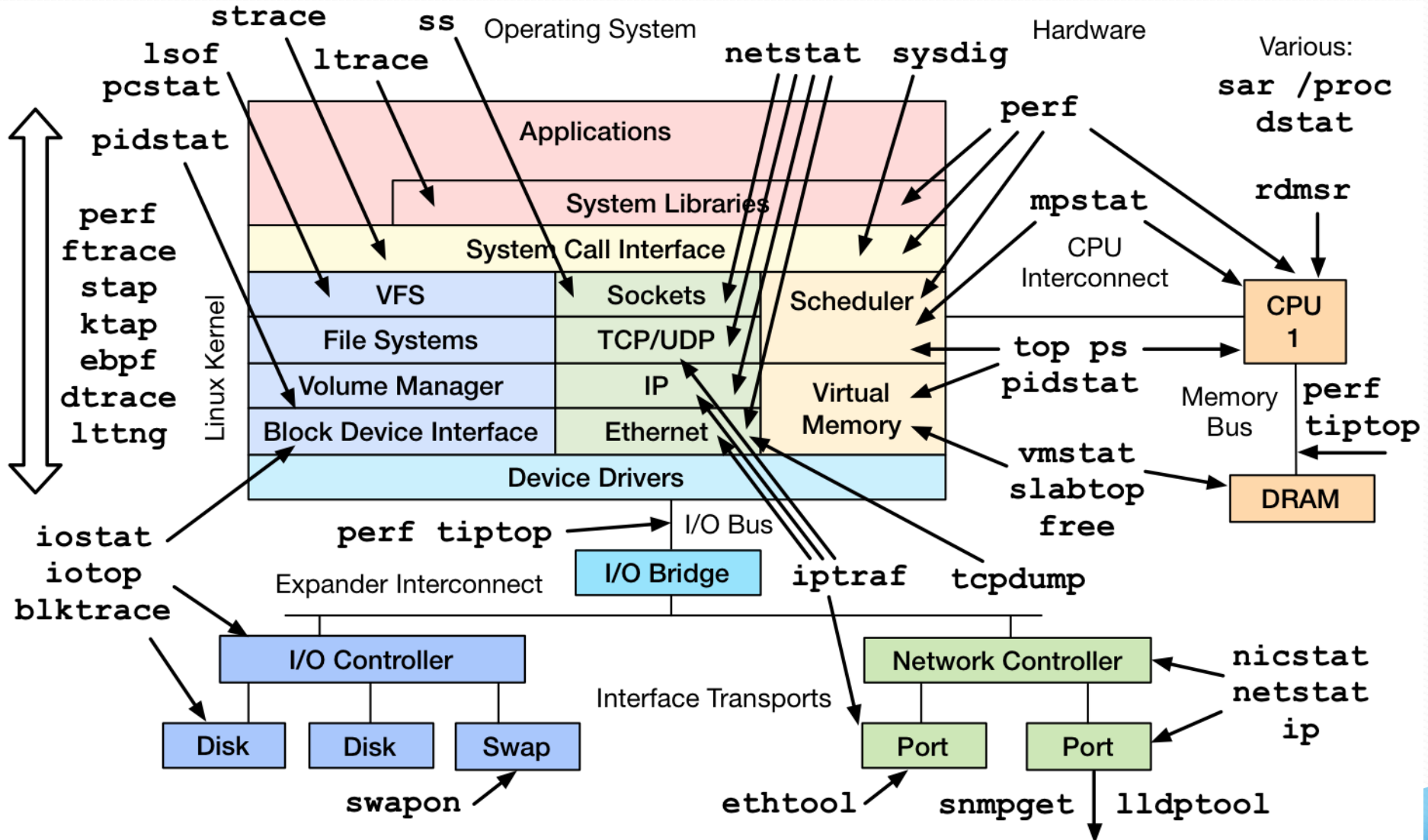
```
$ iostat -xmdz 1
```

```
Linux 3.13.0-29 (db001-eb883efa)    08/18/2014    _x86_64_    (16 CPU)
```

Device:	rrqm/s	wrqm/s	r/s	w/s	rMB/s	wMB/s	\ ...
xvda	0.00	0.00	0.00	0.00	0.00	0.00	/ ...
xvdb	213.00	0.00	15299.00	0.00	338.17	0.00	\ ...
xvdc	129.00	0.00	15271.00	3.00	336.65	0.01	/ ...
md0	0.00	0.00	31082.00	3.00	678.45	0.01	\ ...

... \	avgqu-sz	await	r_await	w_await	svctm	%util
... /	0.00	0.00	0.00	0.00	0.00	0.00
... \	126.09	8.22	8.22	0.00	0.06	86.40
... /	99.31	6.47	6.47	0.00	0.06	86.00
... \	0.00	0.00	0.00	0.00	0.00	0.00

Otras herramientas para monitorización



Procedimiento sistemático de monitorización: método USE (Utilization, Saturation, Errors)

- Para cada recurso (CPU, memoria, E/S, red,...) comprobamos:
 - Utilización: Tanto por ciento de utilización del recurso (tiempo de CPU, tamaño de memoria, ancho de banda, etc.)
 - Saturación: Tamaño de las colas de aquellas tareas que quieren hacer uso de ese recurso (en el caso de la memoria, cantidad de *swapping* a disco).
 - Errores: Mensajes de error del kernel sobre el uso de dichos recursos.
- Ejemplo: CPU
 - Utilización: `vmstat 1, "us" + "sy" + "st"; sar -u`, sumando todos los campos excepto "%idle" e "%iowait"; ...
 - Saturación: `vmstat 1, "r" > nº de CPU; sar -q, "runq-sz" > nº de CPU;...`
 - Errores: usando *perf* si la opción de “eventos de errores específicos de CPU” está habilitada; ...
- Más ejemplos: <http://www.brendangregg.com/USEmethod/use-linux.html>

Otras herramientas para monitorización (II)

- CollectL
 - <http://collectl.sourceforge.net/>. Parecido a sar. Es capaz de ejecutarse de forma interactiva o como un servicio/demonio para recopilar datos históricos.
- Nagios
 - <https://www.nagios.org/>. Permite la monitorización y la generación de alarmas de equipos distribuidos en red (tanto recursos HW como servicios de red). Se puede personalizar mediante la programación de plugins propios.
- Otras herramientas que permiten la monitorización de equipos distribuidos en red: Ganglia, Munin, Zabbix, Pandora FMS...



Programa SarCheck (www.sarcheck.com)

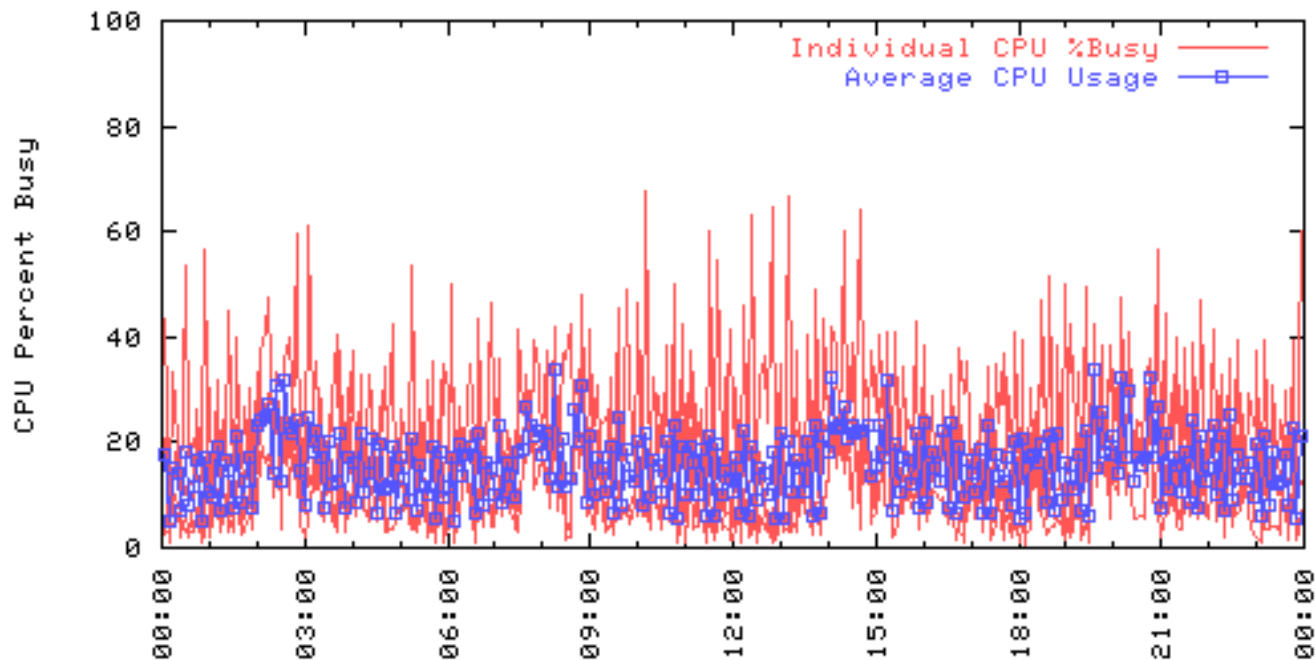
- Herramienta para:
 - Análisis de prestaciones.
 - Sintonización, planificación de la capacidad.
- Genera informes en formato HTML:
 - Estadísticas sobre el uso de recursos.
 - Análisis de recursos, detectando posibles cuellos de botella.
 - Sección de recomendaciones.
- Sistemas Sun Solaris, HP-UX, AIX y Linux x86.
- Basado en el monitor `sar`.
- Utiliza `gnuplot` para generar gráficos.

SarCheckTM

*UNIX Performance Tuning Simplified...
and Linux Performance Tuning too!*

Ejemplo informe generado por SarCheck

Average CPU utilization was only 15.7 percent. This indicates that spare capacity exists within the CPU. If any performance problems were seen during the monitoring period, they were not caused by a lack of CPU power. CPU utilization peaked at 34.00 percent from 08:10:01 to 08:15:01. A CPU upgrade is not recommended because the current CPU had significant unused capacity.



Informe generado por SarCheck (cont.)

Change the virtual memory parameter 'swappiness' from 60 to 70. This controls the systems likelihood of swapping out memory pages which have not been recently used. Larger values will swap out unused pages of memory, speeding the time required to load new pages into memory.

Try to balance the disk I/O load across time or among other disk devices. The load on at least one disk was clearly excessive at peak times. By distributing the load over a greater time period or across other disk devices, the load will not be as likely to cause performance degradation.

Change the bdflush parameter 'nfract' from 50 to 55. This is the percentage of dirty buffers allowed in the buffer cache before the kernel flushes some of them.

Change the bdflush parameter 'nfract_sync' from 80 to 85. This is the percentage of dirty buffers in the buffer cache before the kernel aggressively flushes them synchronously.

To change the value of the bdflush parameters immediately as described in the above recommendations, use the following command:

```
echo "55 500 8 0 500 3000 85 45 0" > /proc/sys/vm/bdflush
```

3.3. Monitorización a nivel de aplicación (profilers)

Monitorización a nivel de aplicación (*profiling*)

- Objetivo
 - Observar el comportamiento de una aplicación concreta con el fin de obtener **información para poder optimizar su código**.
- Información que pueden proporcionar las herramientas de análisis de aplicaciones (*profilers*):
 - ¿Cuánto tiempo tarda en ejecutarse el programa? ¿Qué parte de ese tiempo es de usuario y cuál de sistema? ¿Cuánto tiempo se pierde por las operaciones de E/S?
 - ¿En qué parte del código pasa la mayor parte de su tiempo de ejecución (=hot spots)?
 - ¿Cuántas veces se ejecuta cada línea de programa?
 - ¿Cuántas veces se llama a un procedimiento y desde dónde?
 - ¿Cuánto tiempo tarda en ejecutarse (el código propio de) un procedimiento?
 - ¿Cuántos fallos de caché/página genera cada línea del programa?
 - ...

Una primera aproximación: **time**

El programa `/usr/bin/time` mide el tiempo de ejecución de un programa y muestra algunas estadísticas sobre su ejecución.

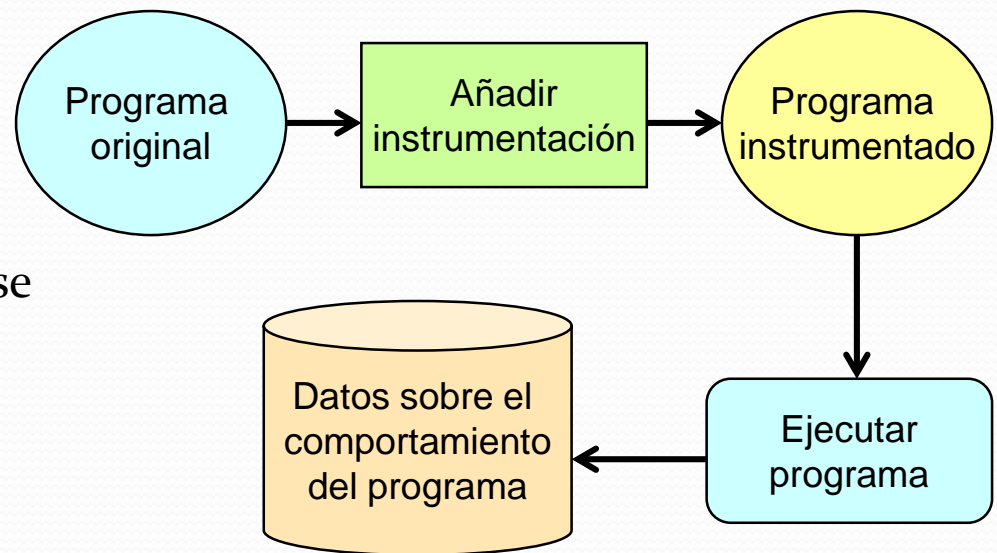
- **real**: tiempo total usado por el sistema (wall-clock CPU time).
- **user**: tiempo de CPU ejecutando en modo usuario.
- **sys**: tiempo de CPU ejecutando código del núcleo.
- Cambios de contexto voluntarios: al tener que esperar a una operación de E/S cede la CPU a otro proceso.
- Cambios involuntarios: Expira su “time slice”.
- Major page faults: requieren acceder al disco.

```
% time ./matr_mult2
real    0m4.862s
user    0m4.841s
sys     0m0.010s

% /usr/bin/time -v ./matr_mult2
User time (seconds): 4.86
System time (seconds): 0.01
Percent of CPU this job got: 99%
Elapsed (wall clock) time 0:04.90
Maximum RSS (kbytes): 48784
Major page faults: 0
Minor page faults: 3076
Voluntary context switches: 1
Involuntary context switches: 195
...
```

Etapas típicas para usar un *profiler*

- Compilar el programa habilitando la recogida de información.
- Ejecutar el programa instrumentado.
 - Ejecución más lenta porque se ha de recoger y dejar la información en un fichero (*profile data*).
- Analizar la información contenida en el fichero de comportamiento.



Monitor gprof

- Da información sobre el tiempo de CPU y número de veces que se ejecuta cada función de un proceso/hilo en un sistema Unix.
- Utilización de gprof para programas escritos en C, C++:
 - Instrumentación en la compilación
 - `gcc prog.c -o prog -pg -g`
 - Ejecución del programa y recogida de información
 - `./prog`
 - La información recogida se deja en el fichero `gmon.out`
 - Visualización de la información referida a la ejecución del programa
 - `gprof prog > prog.gprof`

Utilización del monitor gprof

```
float a=0.3; float b=0.8; float c=0.1;
void main(void) {
    unsigned long i;
    for (i=0;i<80000000;i++) a=a*b/(1+c);
    bucle1(); bucle1(); bucle2();
}
void bucle1(void) {
    unsigned long i;
    for (i=0;i<20000000;i++) {
        c=(c+c*c)/(1+a*c);c=a*b*c; }
    for (i=1;i<=5;i++) bucle3();
}
void bucle2(void) {
    unsigned long i;
    for (i=0;i<5000000;i++) {
        c=(c+c*c+c*c*c)/(1+a*c*c);
        c=a*b*c; }
    for (i=1;i<=8;i++) bucle3();
}
void bucle3(void) {
    unsigned long i;
    for (i=0;i<1000000;i++) {
        c=a*b*c; c=1/(a+b*c);
    }
}
```

- Pasos
 - Instrumentación (-pg) en la compilación
 - Ejecución del programa y recogida de información
 - Obtención de la información referida a la ejecución del programa

```
% gcc bucles.c -pg -o bucles
% ./bucles
% gprof bucles > bucles.prof
```

Salida del monitor gprof

flat profile

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name	
44.47	18.14	18.14				main	← <i>hot spot nº1</i>
34.20	32.08	13.95	2	6.97	8.98	bucle1	← <i>hot spot nº2</i>
17.71	39.30	7.22	18	0.40	0.40	bucle3	
4.32	41.07	1.76	1	1.76	4.97	bucle2	...

- **% time:** Tanto por ciento del tiempo total de ejecución que usa el código propio de la subrutina (código propio es el que pertenece a la subrutina y no a las subrutinas a las que llama).
- **cumulative seconds:** La suma acumulada de los segundos consumidos por la subrutina y por las subrutinas que aparecen encima de ella en la tabla (código propio).
- **self seconds:** tiempo total de ejecución del código propio de la subrutina. Es el criterio por el que se ordena la tabla.
- **self s/call:** tiempo medio de ejecución del código propio por cada llamada a la subrutina.
- **total s/call:** tiempo medio total de ejecución por cada llamada a la subrutina (contando las subrutinas a las que llama)

Salida del monitor gprof

call profile ó
call graph

index	% time	self	children	called	name
[1]	100.0	18.14	22.93		main [1]
		13.95	4.01	2/2	bucle1 [2]
		1.76	3.21	1/1	bucle2 [4]

[2]	43.7	13.95	4.01	2/2	main [1]
		13.95	4.01	2	bucle1 [2]
		4.01	0.00	10/18	bucle3 [3]

		3.21	0.00	8/18	bucle2 [4]
		4.01	0.00	10/18	bucle1 [2]
[3]	17.6	7.22	0.00	18	bucle3 [3]

		1.76	3.21	1/1	main [1]
[4]	12.1	1.76	3.21	1	bucle2 [4]
		3.21	0.00	8/18	bucle3 [3]

bucle1 ha llamado a *bucle3* 10 de las 18 veces que es llamado *bucle3* en total.

Salida del monitor gprof

call profile ó
call graph

index	% time	self	children	called	name
[1]	100.0	18.14	22.93		main [1]
		13.95	4.01	2/2	bucle1 [2]
		1.76	3.21	1/1	bucle2 [4]

[2]	43.7	13.95	4.01	2/2	main [1]
		13.95	4.01	2	bucle1 [2]
		4.01	0.00	10/18	bucle3 [3]

		3.21	0.00	8/18	bucle2 [4]
		4.01	0.00	10/18	bucle1 [2]
[3]	17.6	7.22	0.00	18	bucle3 [3]

		1.76	3.21	1/1	main [1]
[4]	12.1	1.76	3.21	1	bucle2 [4]
		3.21	0.00	8/18	bucle3 [3]

De las 18 veces que *bucle3* es llamado, 8 proceden de *bucle2* y 10 de *bucle1*

Monitor gcov

- Aporta información sobre el número de veces que se ejecuta cada línea de código del programa.
- Utilización de gcov
 - Instrumentación en la compilación
 - `gcc prog.c -o prog -fprofile-arcs -ftest-coverage`
 - Ejecución del programa y recogida de información
 - `./prog`
 - La información recogida se deja en varios ficheros
 - Visualización de la información referida a la ejecución del programa
 - `gcov prog.c` (genera el fichero `prog.c.gcov`)

Salida del monitor gcov

```
%gcc bucles2.c -o bucles2 -fprofile-arcs -ftest-coverage  
%./prog  
%gcov bucles2.c  
100.00% of 16 source lines executed in file bucles2.c  
Creating bucles2.c.gcov
```



bucles.c.gcov

```
void main()  
{  
1   producto();  
1   producto();  
1   producto();  
1   division();  
1   division();  
1   atangente();  
1 }  
  
150000003   producto() {  
150000000   for (i=0;i<500000000;i++)  
3           c=a*b;  
           }  
  
600000002   division() {  
600000000   for (i=0;i<300000000;i++)  
2           c=a/b;  
           }  
  
300000001   atangente() {  
300000000   for (i=0;i<300000000;i++)  
1           c=atan(a);  
           }
```

Otros profilers: Perf

- Perf es un conjunto de herramientas para el análisis de rendimiento en Linux basadas en eventos software y hardware (hacen uso de contadores hardware disponibles en los últimos microprocesadores de Intel y AMD). Permiten analizar el rendimiento de a) un hilo individual, b) un proceso + sus hijos, c) todos los procesos que se ejecutan en una CPU concreta, d) todos los procesos que se ejecutan en el sistema. Algunos de los comandos que proporciona:

```
usage: perf [--version] [--help] COMMAND [ARGS]
```

- **list**: Lista todos los eventos disponibles.
- **stat**: Cuenta el número de eventos.
- **record**: Recolecta muestras cada vez que se produce un determinado conjunto de eventos. Fichero de salida: perf.data.
- **report**: Analiza perf.data y muestra las estadísticas generales.
- **annotate**: Analiza perf.data y muestra los resultados a nivel de código ensamblador y código fuente (si está disponible).

Perf: Algunos tipos de eventos

- **perf list**

• cpu-clock	[Sw]	• LLC-load-misses	[Hw]
• task-clock	[Sw]	• LLC-store-misses	[Hw]
• minor-faults	[Sw]	• dTLB-load-misses	[Hw]
• major-faults	[Sw]	• dTLB-store-misses	[Hw]
• context-switches OR cs	[Sw]	• dTLB-prefetch-misses	[Hw]
• cpu-cycles OR cycles	[Hw]	• iTLB-load-misses	[Hw]
• instructions	[Hw]	• branch-load-misses	[Hw]
• cache-misses	[Hw]	• sched:sched_stat_runtime	[Trace]
• branch-misses	[Hw]	• sched:sched_pi_setprio	[Trace]
• L1-dcache-load-misses	[Hw]	• syscalls:sys_exit_socket	[Trace]
• L1-dcache-store-misses	[Hw]	• [...]	

- La lista completa de eventos hardware depende del tipo concreto de microprocesador.

- Intel: <http://www.intel.com/Assets/PDF/manual/253669.pdf> (cap. 18,19).
- AMD: http://support.amd.com/TechDocs/55072_AMD_Family_15h_Models_70h-7Fh_BKDG.pdf (sección 3.24).

Perf: Ejemplos de uso como profiler

- Ejemplo 1: Cuento el número ciclos, instrucciones, cambios de contexto y fallos de página del proceso ``noploop 1``. Repito el experimento 10 veces:

```
$ perf stat -e task-clock,cycles,context-switches,page-faults,instructions -r 10 -- noploop 1
```

```
Performance counter stats for 'noploop 1' (10 runs):
```

1000.308987 task-clock	#	1.000 CPUs utilized	(+- 0.00%)
3,729,474,643 cycles	#	3.728 GHz	(+- 0.11%)
2 context-switches	#	0.002 K/sec	(+- 8.55%)
110 page-faults	#	0.110 K/sec	(+- 0.09%)
3,594,555,795 instructions	#	0.96 insns per cycle	(+- 0.39%)
1.000582770 seconds time elapsed			(+- 0.00%)

- Ejemplo 2: Recolecto muestras del programa `./naive` cada cierto número de ocurrencias (por defecto 4000) de los eventos: `cpu-clock` y `faults`. Guardo la información en `perf.data`.

```
> perf record -e cpu-clock,faults ./naive
```

```
[ perf record: Woken up 14 times to write data ]
```

```
[ perf record: Captured and wrote 3.438 MB perf.data (~150220 samples) ]
```

Perf: Ejemplos de uso como profiler (II)

- Ejemplo 3: Muestro las funciones de mi programa que más ciclos de reloj consumen y las que más fallos de página provocan:

```
> perf report --stdio --dsos=naive,libc-2.13.so
```

```
# cmdline : /usr/bin/perf_3.6 record -e cpu-clock,faults ./naive
```

```
# Samples: 74K of event 'cpu-clock'
```

```
# Event count (approx.): 74813
```

# Overhead	Command	Shared Object	Symbol
99.32%	naive	naive	[.] multiply_matrices
0.34%	naive	libc-2.13.so	[.] random
0.15%	naive	naive	[.] initialize_matrices

```
...
```

```
# Samples: 63 of event 'page-faults'
```

```
# Event count (approx.): 807
```

# Overhead	Command	Shared Object	Symbol
91.82%	naive	naive	[.] initialize_matrices
6.44%	naive	libc-2.13.so	[.] 0x000ca168

```
...
```

Perf: Ejemplos de uso como profiler (III)

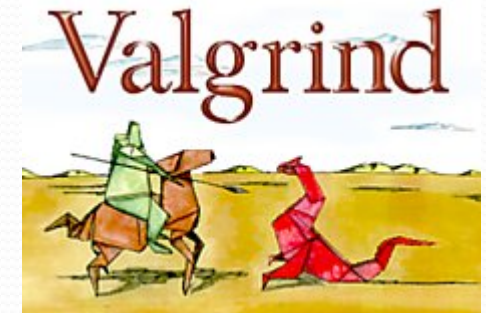
- Ejemplo 4: Muestro las líneas concretas de mi programa que más ciclos de reloj consumen:

```
> perf annotate --stdio --dsos=naive --symbol=multiply_matrices
```

```
Percent |      Source code & Disassembly of naive
        :      void multiply_matrices()
        :
        :      ...
        :      for (i = 0 ; i < MSIZE ; i++) {
        :      for (j = 0 ; j < MSIZE ; j++) {
        :      float sum = 0.0 ;
        :      for (k = 0 ; k < MSIZE ; k++) {
1.67    :      8c18:  add    r3, r3, #1
54.13   :      8c1c:  cmp    r3, #500      ; 0x1f4
        :      sum = sum + (matrix_a[i][k] * matrix_b[k][j]) ;
2.39    :      8c20:  mov    r9, r1
36.60   :      8c24:  vmla.f32    s15, s13, s14
        :
        :      ...
```


Otros profilers: Valgrind

- Valgrind es un conjunto de herramientas para el análisis y mejora del código. Entre éstas, encontramos:
 - Callgrind, una versión más refinada de gprof.
 - Cachegrind, un profiler de la memoria caché.
 - Memcheck, un detector de errores de memoria.
- Valgrind puede analizar cualquier binario ya compilado (no necesita instrumentar el programa a partir de su código fuente). Valgrind actúa, esencialmente, como una máquina virtual que emula la ejecución de un programa ejecutable en un entorno aislado.
- Como desventaja, el sobrecoste computacional es muy alto. La emulación del programa ejecutable puede tardar decenas de veces más que la ejecución directa del programa de forma nativa.



Otros profilers: V-Tune y CodeXL

- Al igual que Perf pueden hacer uso tanto de eventos software como hardware. Ambos programas funcionan tanto para Windows como para Linux, y permiten obtener información sobre los fallos de caché, fallos de TLB, bloqueos/rupturas del cauce, fallos en la predicción de saltos, cerrojos y esperas entre hebras, etc. asociados a cada línea del programa (tanto en código fuente como en código ensamblador).
- También se pueden usar como depuradores (debuggers), permiten la ejecución remota y son capaces de medir también el rendimiento de GPU, controlador de memoria, conexiones internas a la CPU, etc.

