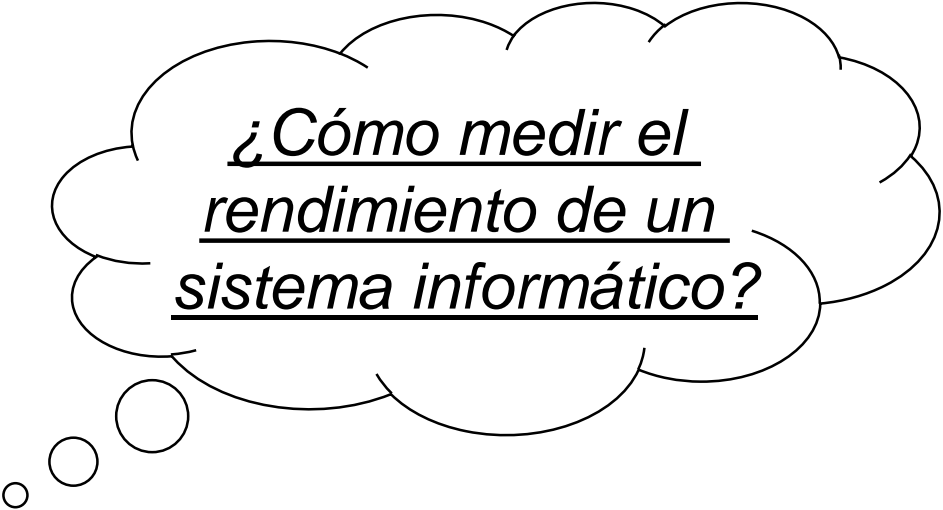


Monitorización de sistemas y programas



¿Cómo medir el
rendimiento de un
sistema informático?

Usuarios, administradores y diseñadores

Contenido

1. Introducción

Medida por detección de eventos

Medida por muestreo

2. Monitores de actividad

Monitores *software*, *hardware* e híbridos

3. Monitorización en Unix

Carga media de un sistema

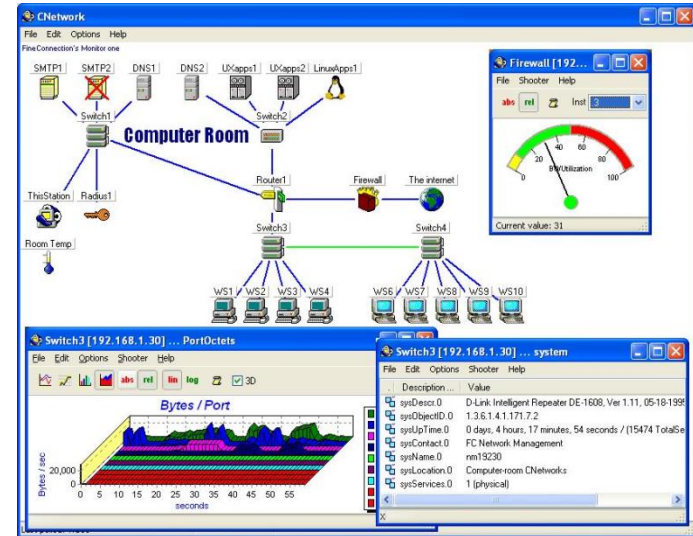
Herramientas de monitorización

Análisis con SarCheck[®]

4. Análisis de programas (*profiling*) en sistemas Unix

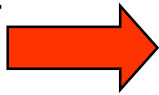
Análisis por funciones: `gprof`

Análisis por líneas: `gcov`

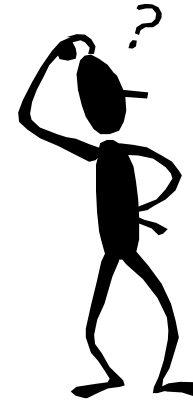


La medida

Carga



¿Qué está
ocurriendo?



⇒ Problema de la medida

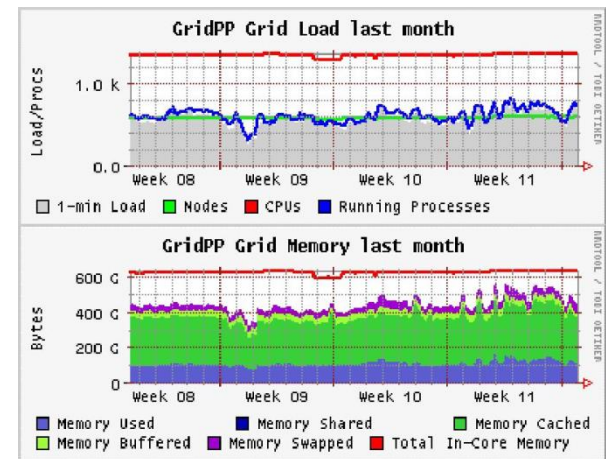
- ¿Qué información?
- ¿Cómo se puede extraer y dónde grabarla?

⇒ El instrumento de medida puede perturbar el funcionamiento del sistema

Técnicas de medida

⇒ ¿Cómo se toman medidas del sistema?

- Cada vez que ocurre un evento
- Cada periodo fijo de tiempo



Medida por muestreo

Medida por detección
de eventos

Detección de eventos

⇒ Estado del sistema

- Contenido de todas las memorias

⇒ Evento

- Provoca un cambio del estado

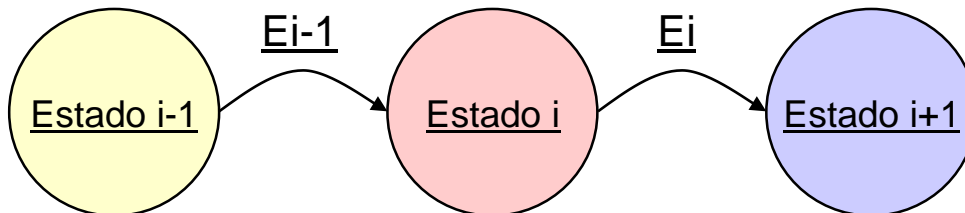
⇒ Volumen de información recogida

- Depende de la frecuencia de los eventos

⇒ Ejemplos de eventos:

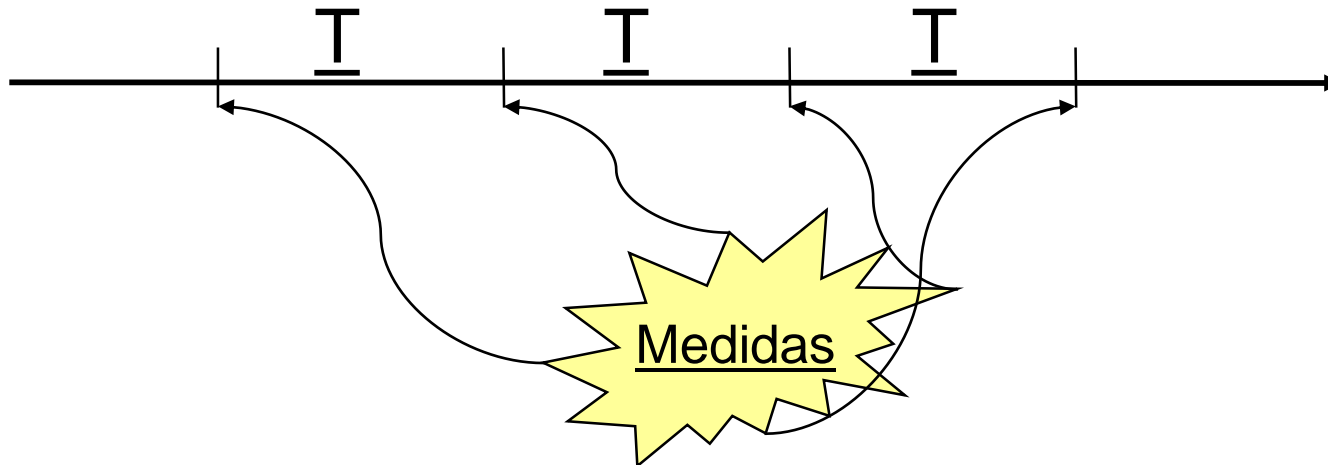
- Inicio/fin de la ejecución de un programa
- Activación de las señales RD* y WR* de memoria
- Acierto/fallo en memoria cache
- Atención a un dispositivo periférico
- Abrir/cerrar un fichero

⇒ Una gran parte de los eventos (no todos) pueden ser detectados por software



Muestreo

- ⇒ Observación a intervalos regulares o aleatorios
- Análisis estadístico de datos más fácil
 - Volumen de información recogida y precisión: dependen de T

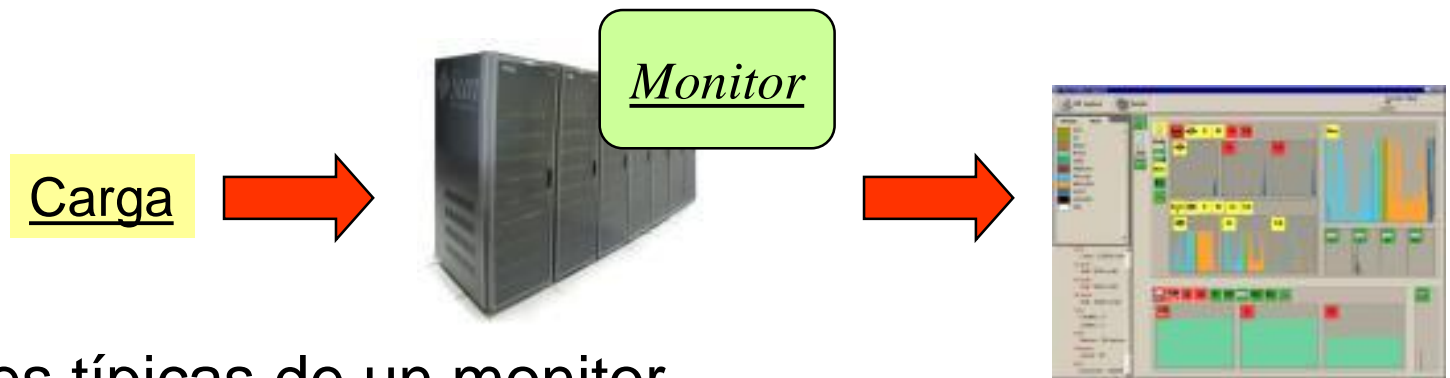


2. Monitores de actividad

Monitores software, hardware e híbridos

Concepto de monitor

⇒ Herramienta diseñada para observar la actividad de un sistema informático mientras es utilizado por los usuarios



⇒ Acciones típicas de un monitor

- Observar el comportamiento
- Recoger datos estadísticos
- Analizar estos datos
- Mostrar los resultados

Utilidad de los monitores

⇒ Administrador

- Conocer la utilización de los recursos (detección de cuellos de botella)
- Ajustar los parámetros del sistema (Entonar)

⇒ Analista

- Parametrizar la carga real.
- Calcular los parámetros de entrada a modelos (analíticos o simulación).

⇒ Sistema

- Adaptarse dinámicamente a la carga

⇒ Desarrollo de aplicaciones: entender su funcionamiento: entonarlas.



Atributos de los monitores

⇒ Interferencia o sobrecarga
(*overhead*)

- Diferentes grados de intrusismo

⇒ Precisión

- Calidad de la medida

⇒ Resolución

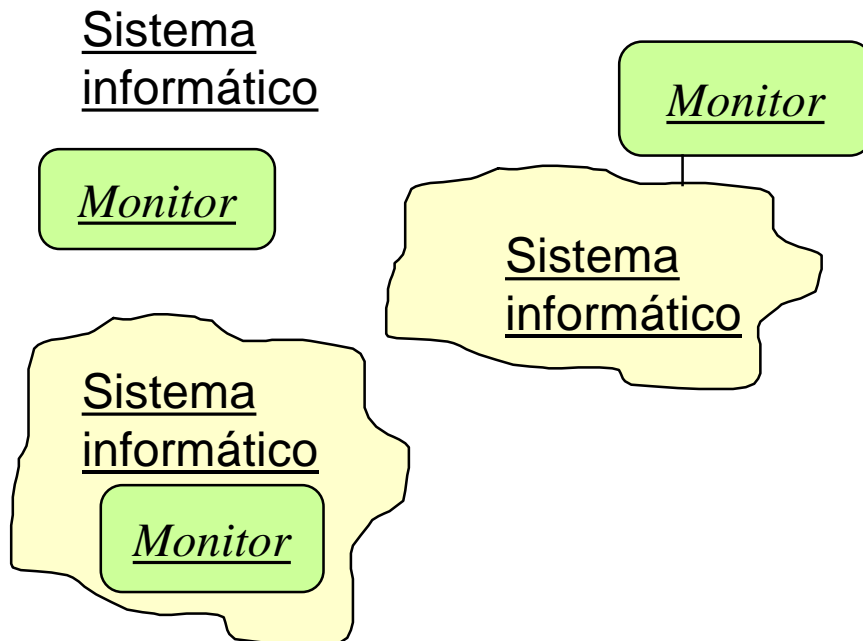
- Frecuencia de medida

⇒ Ámbito o dominio de medida

- Qué mide

⇒ Coste

⇒ Facilidad de instalación y uso



Implementación de los monitores

⇒ Software

- Programas instalados en el sistema

⇒ Hardware

- Dispositivos externos al sistema

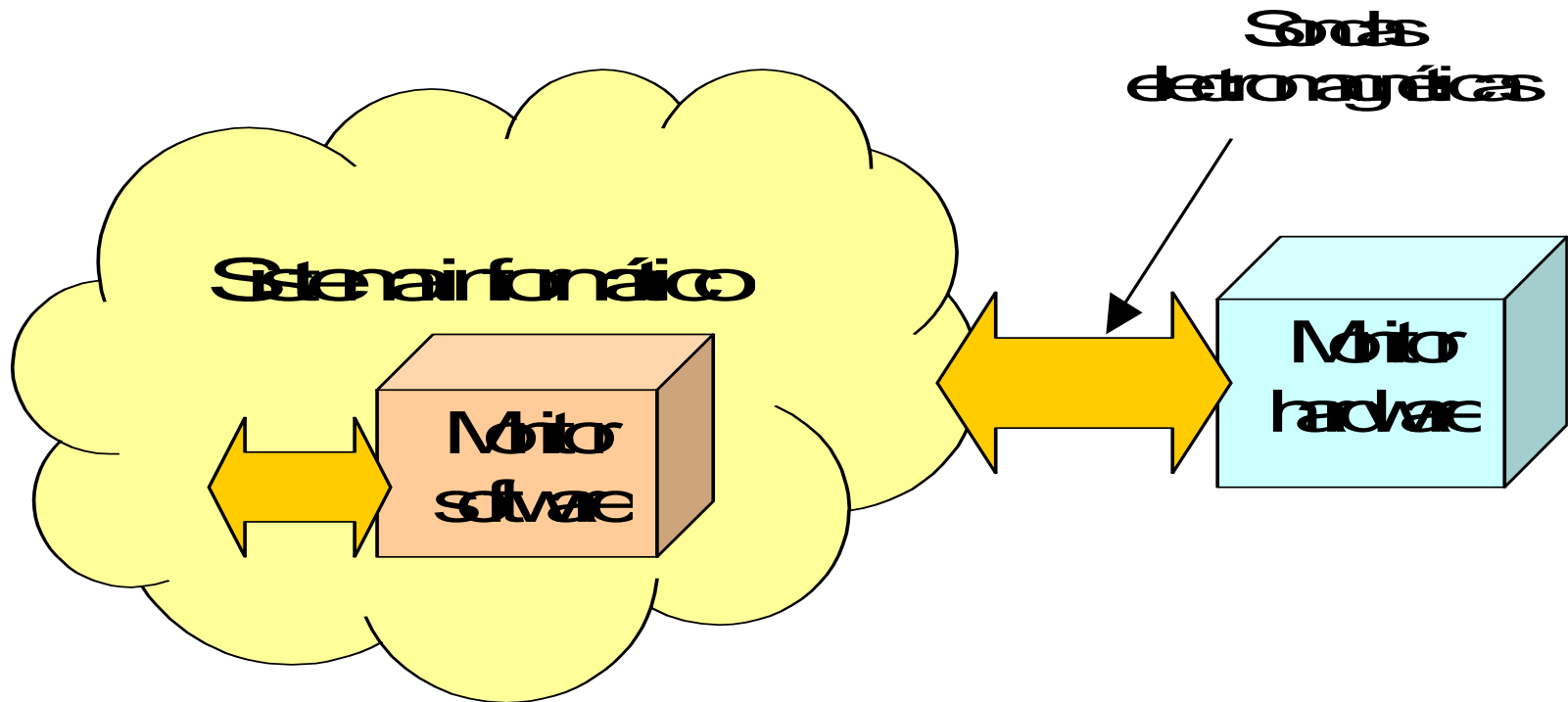
⇒ Híbridos

- Utiliza los dos tipos anteriores

Los más
habituales

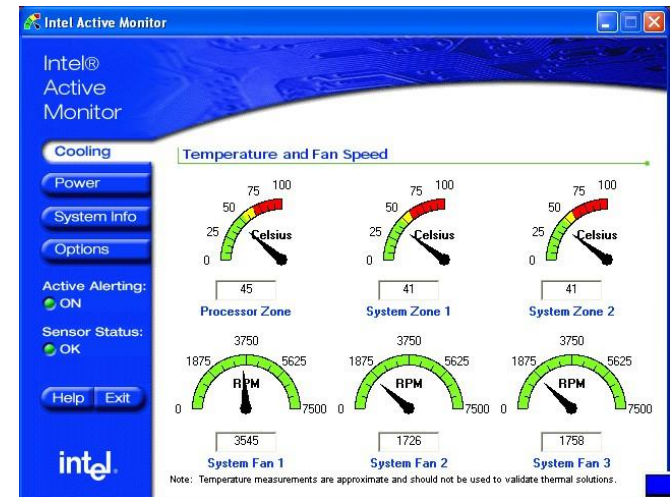
Entornos
muy
específicos

Situación de los monitores



Monitores software

- ⇒ Son los más usados
- ⇒ Activación → Ejecución de instrucciones → Sobrecarga
- ⇒ Implementación
 - Adición de un nuevo programa
 - Modificación del software a medir
 - Modificación del sistema operativo



Monitores hardware

- ⇒ Instrumentos independientes (externos) del sistema a monitorizar conectados a este mediante sondas electromagnéticas
- ⇒ Ventajas
 - No usan recursos del sistema monitorizado
 - Rapidez (circuitos electrónicos)
- ⇒ Inconvenientes
 - Los sistemas no facilitan la instalación de sondas
 - Personal especializado para su operación
 - Hay magnitudes no accesibles por hardware
 - Costosos



Monitores: clasificación

De acuerdo a cómo organizan la información:

- ⇒ **Orientados a clases:** realizan medidas a nivel de programa.
- ⇒ **Orientados a recursos:** organizan la información por recursos (utilizaciones, tiempos de servicio, longitudes media de cola, etc) y no están al tanto de qué procesos usan tales recursos ni en qué medida.

3. Monitorización en Unix

Herramientas de medida:


time, who, w, uptime, ps, top, vmstat,
df, du, hdparm, sar, mpstat, iostat

time

⇒ Mide el tiempo de ejecución de un programa

– Refleja la percepción de las prestaciones del sistema por parte del usuario

- real: tiempo total usado por el sistema (tiempo de respuesta)
- user: tiempo de CPU ejecutando en modo usuario (user-state CPU time)
- sys: tiempo de CPU en modo supervisor (system-state CPU time) ejecutando código del núcleo



<u>%time quicksort</u>		
<u>real</u>	<u>6m</u>	<u>23s</u>
<u>user</u>	<u>3m</u>	<u>50s</u>
<u>sys</u>	<u>2m</u>	<u>10s</u>

Tiempo de respuesta = real = 383 s

Tiempo de CPU = user+sys = 360 s (94% del total)

Tiempo de espera = real-(user+sys) = 23 s (6% del total)

consumido en espera de I/O o en la ejecución de otros programas

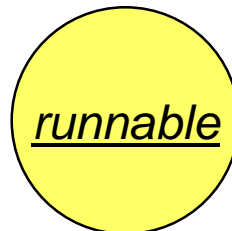
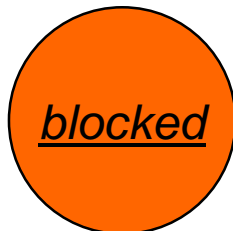
Carga media del sistema Unix

⇒ Estados básicos de un proceso

- En ejecución (*running process*)
- En espera
 - Dispone de todo menos de la CPU (*runnable process*)
 - Bloqueado en una operación de I/O (*blocked process*)

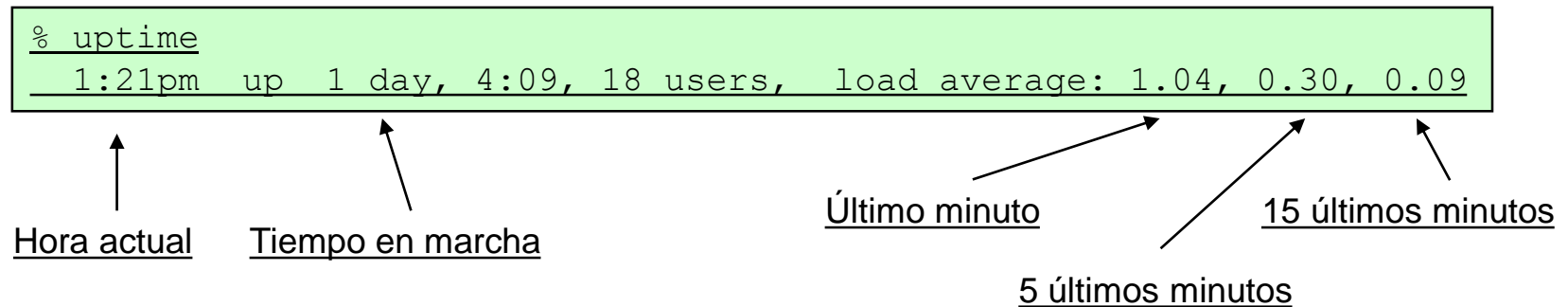
⇒ La cola de procesos del núcleo (*run queue*) está formada por aquellos que pueden ejecutarse (*runnable*)

⇒ Carga media (*system load average*): número medio de procesos en ejecución y en la cola del núcleo



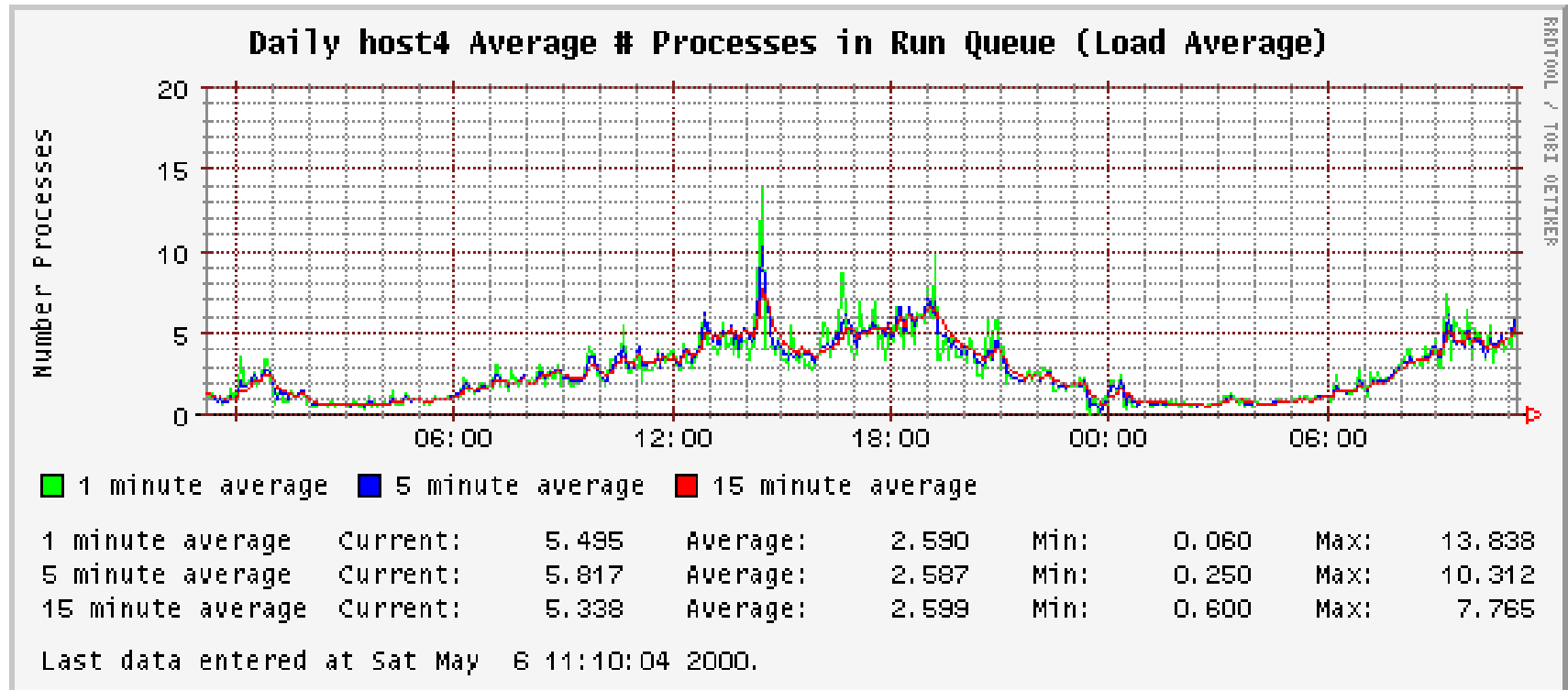
uptime

⇒ Tiempo que lleva el sistema en marcha y la carga media que soporta



- Estimación de la carga
 - Operación normal: hasta 3
 - Muy alta: entre 4 y 7
 - Excesivamente alta: mayor que 10
- La carga se tolera según la configuración de cada sistema

Evolución típica de la carga media



ps (process status)

⇒ Información sobre el estado de los procesos del sistema

- Es una de las herramientas más importantes empleadas en tareas de monitorización
- Tiene una gran cantidad de parámetros

```
$ 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
xavier	30023	0.0	0.5	2464	1492	pts/0	R	09:27	0:00	ps aur

Información aportada por ps

- USER
 - Usuario que lanzó el proceso
- %CPU, %MEM
 - Porcentaje de procesador y memoria física usada
- SIZE (o VSIZE)
 - Memoria (KB) de datos (no código) ocupada por el proceso (*non shared virtual memory*)
- RSS (*resident size*)
 - Memoria (KB) física ocupada por el proceso
- STAT
 - R (*runnable*), T (*stopped*), P (*waiting for page-in*), D (*waiting for disk I/O*), S (*sleeping for less than 20 s*), I (*idle for more than 20 s*), Z (*zombie: terminated but not died*)
 - W (*swapped out*), > (*memory soft limit exceeded*)
 - N (*running niced*), < (*high niced level*)

top

⇒ Carga media, procesos, consumo de memoria

⇒ Se actualiza dinámicamente

```
8:48am up 70 days, 21:36, 1 user, load average: 0.28, 0.06, 0.02
47 processes: 44 sleeping, 3 running, 0 zombie, 0 stopped
CPU states: 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	PRI	NI	SIZE	RSS	SHARE	STAT	LC	%CPU	%MEM	TIME	COMMAND
9826	carlos	0	0	388	388	308	R	0	99.6	0.1	0:22	simulador
9831	miguel	19	0	976	976	776	R	0	0.3	0.3	0:00	top
1	root	20	0	76	64	44	S	0	0.0	0.0	0:03	init
2	root	20	0	0	0	0	SW	0	0.0	0.0	0:00	keventd
4	root	20	19	0	0	0	SWN	0	0.0	0.0	0:00	ksoftiq
5	root	20	0	0	0	0	SW	0	0.0	0.0	0:13	kswapd
6	root	2	0	0	0	0	SW	0	0.0	0.0	0:00	bdfush
7	root	20	0	0	0	0	SW	0	0.0	0.0	0:10	kdated
8	root	20	0	0	0	0	SW	0	0.0	0.0	0:01	kinoded
11	root	0	-20	0	0	0	SW<	0	0.0	0.0	0:00	recoved

vmstat (virtual memory statistics)

⇒ Paging (paginación), swapping, interrupciones, cpu

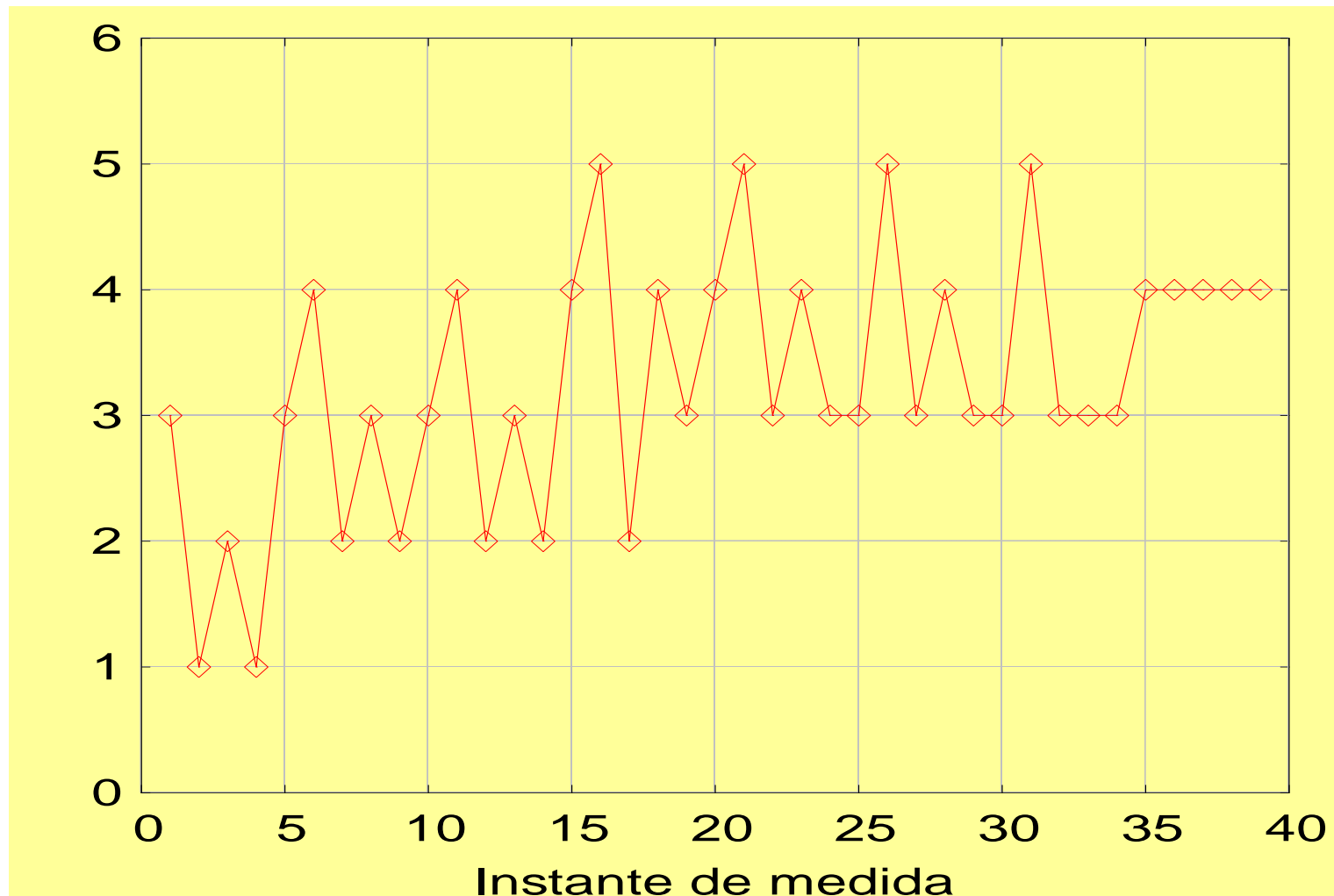
- La primera línea no sirve para nada

```
% vmstat -n 1 6
```

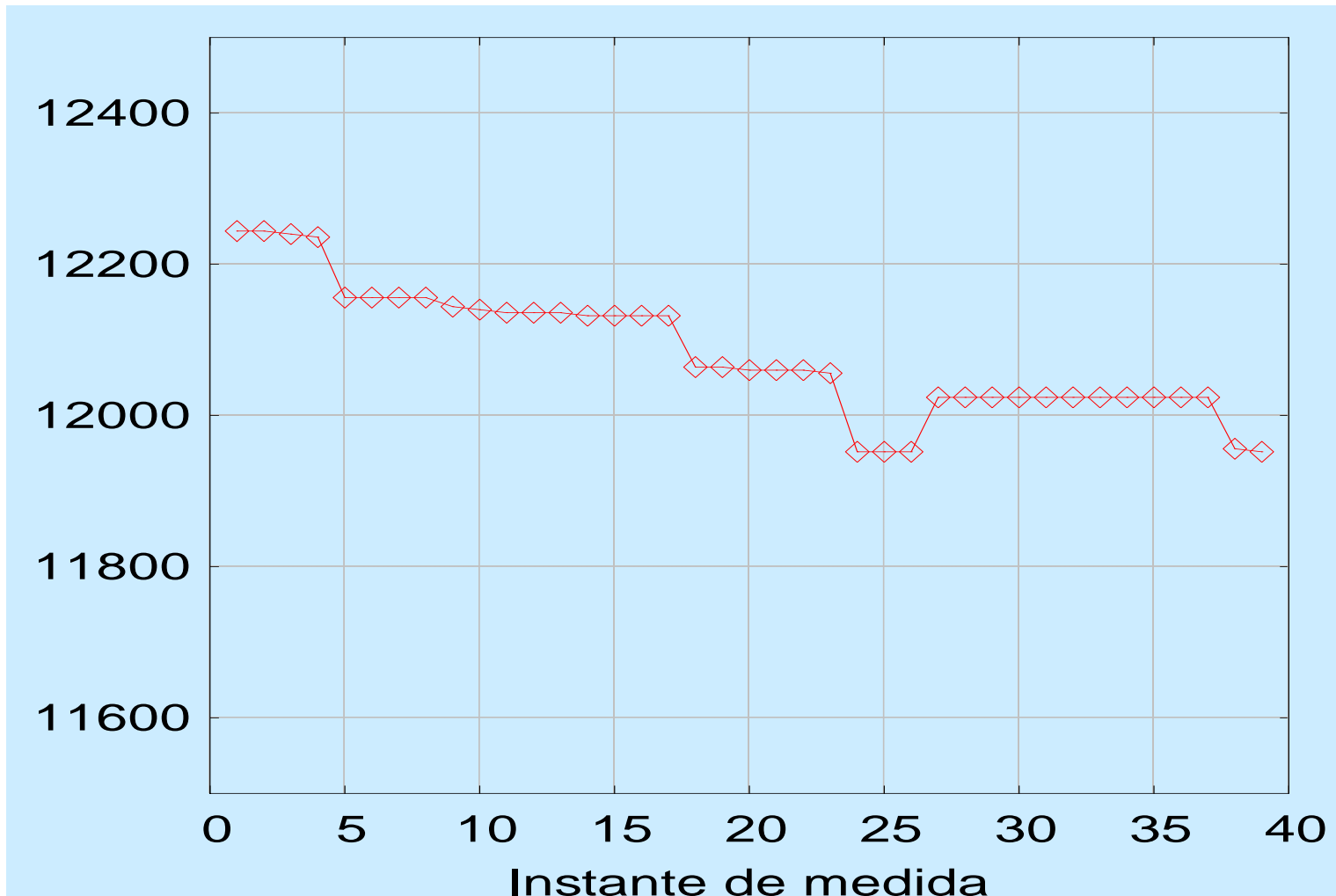
procs			memory				swap		io		system			cpu	
r	b	w	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id
0	0	0	868	8964	60140	342748	0	0	23	7	222	199	1	4	95
0	0	0	868	8964	60140	342748	0	0	0	14	283	278	0	7	93
0	0	0	868	8964	60140	342748	0	0	0	0	218	212	6	2	93
0	0	0	868	8964	60140	342748	0	0	0	0	175	166	3	3	94
0	0	0	868	8964	60140	342752	0	0	0	2	182	196	0	7	93
0	0	0	868	8968	60140	342748	0	0	0	18	168	175	3	8	89

- Procesos: r (runnable), b (I/O blocked), w (swapped out)
- Bloques por segundo transmitidos: bi (blocks in), (blocks out)
- KB/s entre memoria y disco: si (swapped in), so (swapped out)
- in (interrupts por second), cs (context switches)

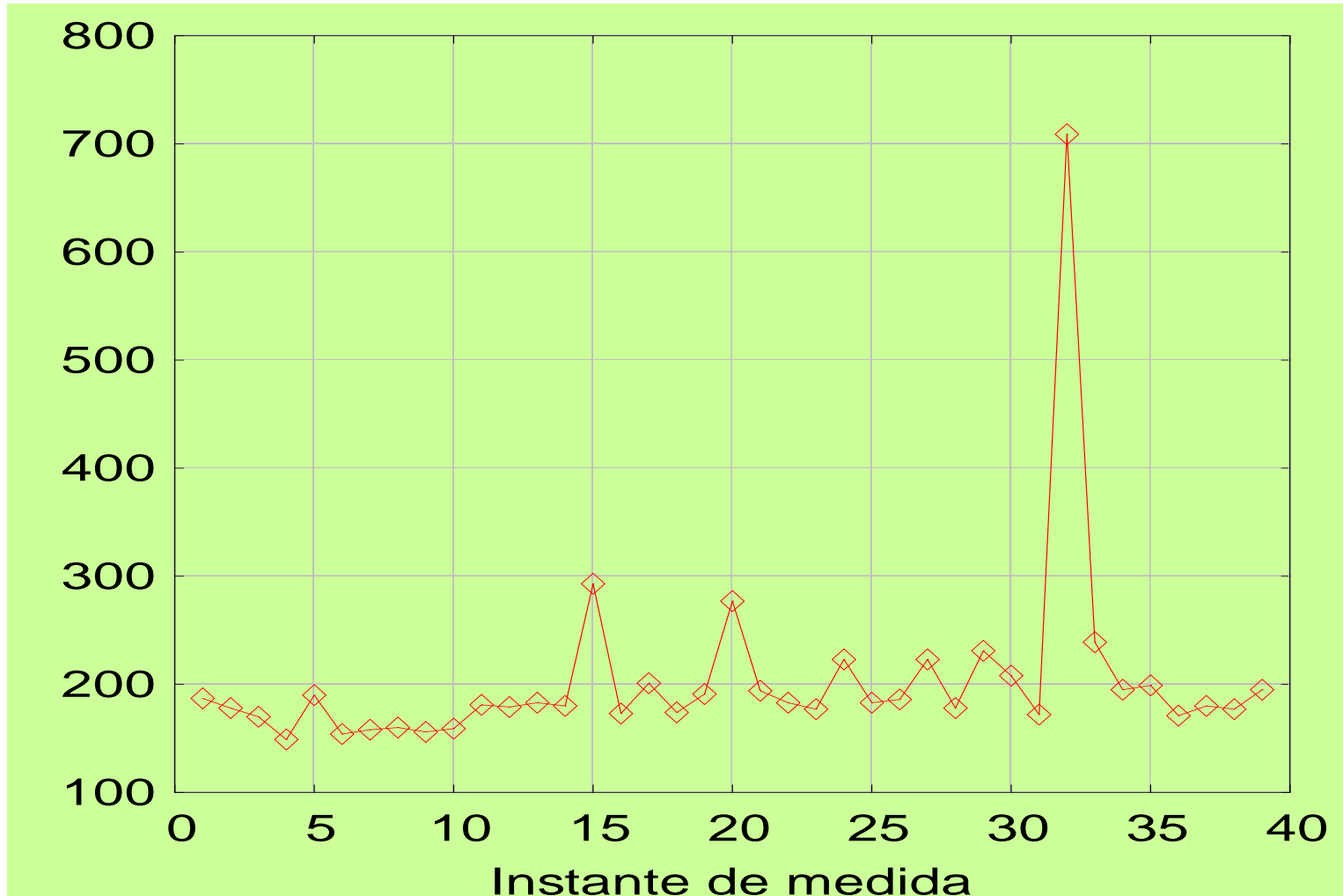
Procesos disponibles para ejecutar



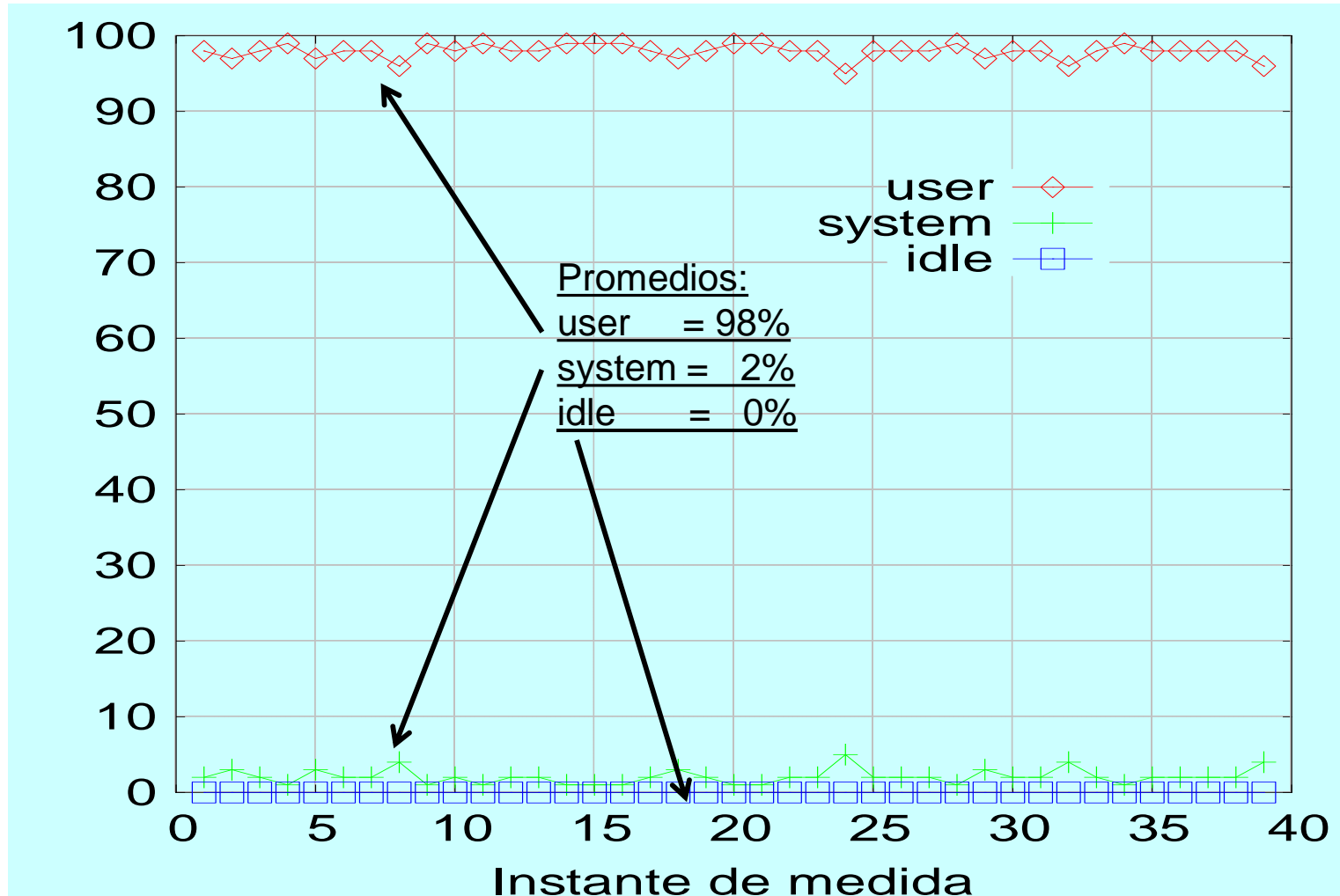
Capacidad de memoria libre



Interrupciones por segundo



Utilización del procesador



Información sobre los discos

⇒ df (*filesystem disk space usage*)

```
$ df
Filesystem            1k-blocks      Used Available Use% Mounted on
/dev/hda2              9606112    3017324    6100816   34% /
/dev/hdb1             12775180    9236405    3140445   75% /home
```

⇒ du (*file space usage*)

```
$ du doc
160      doc/cartas
432      doc
```

⇒ hdparm (*hard disk parameters*)

```
$ hdparm -g /dev/hda
/dev/hda:
geometry      = 790/255/63, sectors = 12706470, start = 0
```

```
$ hdparm -tT /dev/hda
Timing buffer-cache reads: 128 MB in 1.15 seconds =111.30 MB/sec
Timing buffered disk reads: 64 MB in 6.04 seconds = 10.60 MB/sec
```


El directorio /proc

⇒ Contiene ficheros con información del sistema

- Configuración
- Estadísticas: contadores

⇒ Ejemplos

- cpuinfo
- meminfo
- interrupts
- devices
- etc.



```
%more /proc/cpuinfo
processor          : 0
vendor_id         : GenuineIntel
cpu family        : 6
modelo            : 5
modelo name       : Pentium II (Deschutes)
stepping          : 2
cpu MHz           : 350.807487
cache size        : 512 KB
fdiv_bug          : no
fpu               : yes
fpu_exception     : yes
cpuid level       : 2
wp               : yes
flags             : fpu vme de pse tsc msr
bogomips          : 349.80
```

El monitor sar

⇒ sar (system activity reporter)

- Muy utilizado por los administradores de sistemas Unix en la detección de cuellos de botella (bottlenecks)
- Información sobre todo el sistema
 - Actual: qué está pasando el día de hoy, o ahora mismo, al sistema
 - Histórica: qué ha pasado en el sistema en otros días pasados
 - Ficheros históricos
 - saDD, donde los dígitos DD indican el día del mes
- Hace uso de contadores estadísticos del núcleo del sistema operativo ubicados en los directorios /proc y /dev/kmem

⇒ Disponibilidad en internet

- <http://perso.orange.fr/sebastien.godard>
- <ftp://atcomputing.nl/pub/tools/linux>

Ejemplo de salidas del monitor sar

⇒ Utilización de los procesadores (sistema biprocesador)

```
$ sar
00:00:00      CPU      %user      %nice      %system      %idle
00:05:00      all       0.09       0.00       0.08       99.83
00:10:00      all       0.01       0.00       0.01       99.98
...
11:15:00      all       0.02       0.00       0.02       99.96
11:20:00      all       0.44       0.00       0.20       99.36
11:25:00      all       0.05       0.00       0.02       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°
```


Otras herramientas de S. Godard

<http://www.tecmint.com/sysstat-commands-to-monitor-linux/>

⇒ mpstat (*processors related statistics*)

```
$ mpstat -P 1 3 5
12:07:03      CPU      %user      %nice %system      %idle      intr/s
12:07:06          1    100.00        0.00      0.00      0.00        63.00
12:07:09          1    100.00        0.00      0.00      0.00        66.00
12:07:12          1    100.00        0.00      0.00      0.00        44.00
12:07:15          1    100.00        0.00      0.00      0.00        74.00
12:07:18          1    100.00        0.00      0.00      0.00        50.00
Average:          1    100.00        0.00      0.00      0.00        59.40
```

⇒ iostat (*input/output statistics*)

```
$ iostat
cpu-avg:   %user   %nice   %sys   %idle
           3.70    0.02    0.48   95.81

Device:            tps    Blk_read/s    Blk_wrtn/s    Blk_read    Blk_wrtn
dev2-0              0.00         0.00         0.00        133         0
dev3-0              0.55         4.53         6.62    11726226    17108122
dev3-1              0.01         0.00         0.61      2698     1590072
```

4. Análisis de programas (*profiling*) en sistemas Unix

Análisis por funciones: `gprof`

Análisis por líneas de código: `gcov`

Análisis de programas

⇒ Objetivo

- Observar el comportamiento de los programas

⇒ Información que proporcionan las herramientas de análisis

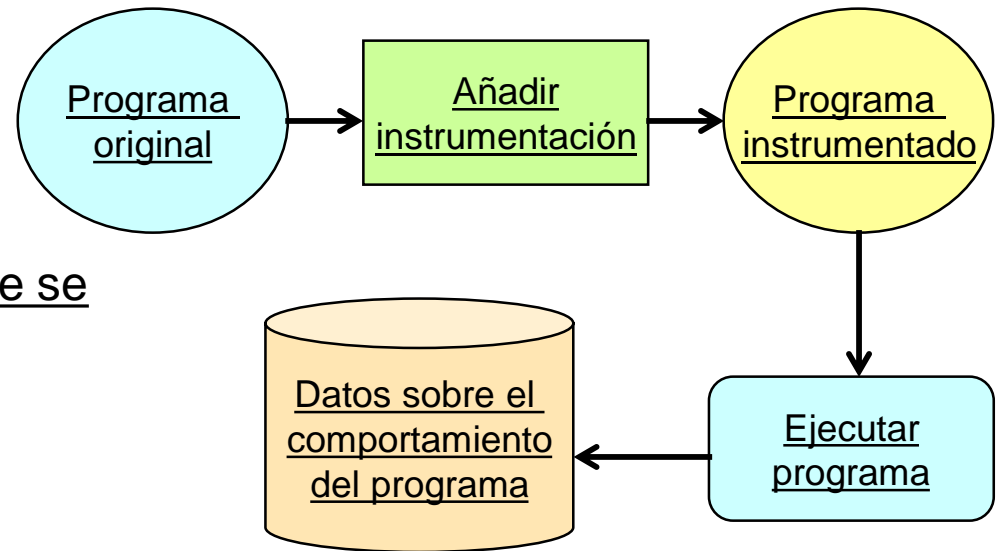
- ¿Dónde pasa la mayor parte de su tiempo de ejecución?
- ¿Cuántas veces se ejecuta una línea de programa?
- ¿Cuántas veces se llama a un procedimiento y desde dónde?
- ¿Qué funciones se llaman desde un determinado procedimiento?

⇒ Herramientas disponibles en Unix

- Orden `gprof`: orientada al análisis de procedimientos
- Orden `gcov`: orientada al análisis de líneas y bloques de instrucciones

Etapas a seguir

- ⇒ 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 ejecución y número de veces que se ejecuta una función
- ⇒ Utilización de gprof
 - Instrumentación en la compilación
 - gcc prog.c -o prog -pg -g -a
 - 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

⇒ Pasos

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

```
#include <stdio.h>
#include <time.h>
#include <math.h>
double a=3.14,b=6.34,c=-3.03;
long y;

void main()
{
    producto(); producto(); producto();
    division(); division();
    atangente();
}

producto()
{for (i=0; i<50000000; y++) c=a*b;}

division()
{for (i=0; i<30000000; y++) c=a/b;}

atangente()
{for (i=0; i<30000000; y++) c=atan(a);}
```

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

Salida del monitor gprof

Each sample counts as 0.01 seconds

flat profile

<u>%</u>	<u>cumulative</u>	<u>self</u>		<u>self</u>	<u>total</u>	
<u>time</u>	<u>seconds</u>	<u>seconds</u>	<u>calls</u>	<u>ms/call</u>	<u>ms/call</u>	<u>name</u>
62.79	11.12	11.12	2	5560.00	5560.00	division
20.33	14.72	3.60	1	3600.00	3600.00	atangente
16.88	17.71	2.99	3	996.67	996.67	producto

<u>index</u>	<u>% time</u>	<u>self</u>	<u>children</u>	<u>called</u>	<u>name</u>	
[1]	100.0	0.00	17.71		main	[1]
		11.12	0.00	2/2	division	[2]
		3.60	0.00	1/1	atangente	[3]
		2.99	0.00	3/3	producto	[4]

		11.12	0.00	2/2	main	[1]
[2]	62.8	11.12	0.00	2	division	[2]

		3.60	0.00	1/1	main	[1]
[3]	20.3	3.60	0.00	1	atangente	[3]

		2.99	0.00	3/3	main	[1]
[4]	16.9	2.99	0.00	3	producto	[4]

call profile

GetRusage

⇒ Ejemplo

```
struct rusage {  
struct timeval ru_utime; /* user time used */  
struct timeval ru_stime; /* system time used */  
long ru_maxrss; /* maximum resident set size */  
long ru_ixrss; /* integral shared memory size */  
long ru_idrss; /* integral unshared data size */  
long ru_isrss; /* integral unshared stack size */  
long ru_minflt; /* page reclaims */  
long ru_majflt; /* page faults */  
long ru_nswap; /* swaps */  
long ru_inblock; /* block input operations */  
long ru_oublock; /* block output operations */  
long ru_msgsnd; /* messages sent */  
long ru_msgrcv; /* messages received */  
long ru_nsignals; /* signals received */  
long ru_nvcsw; /* voluntary context switches */  
long ru_nivcsw; /* involuntary context switches */  
};
```


Gettimeofday

```
#include <stdio.h>  
#include <sys/time.h>  
void main() {  
struct timeval t_start, t_finish;  
double elapsed;  
gettimeofday(&t_start, NULL);  
/* do some work here */  
gettimeofday(&t_finish, NULL);  
elapsed = t_finish.tv_sec - t_start.tv_sec +  
(t_finish.tv_usec - t_start.tv_usec) / 1.e6;  
printf("Elapsed time %.9f seconds\n", elapsed); }
```