Ingeniería de Servidores

Apuntes de Teoría

INGENIERÍA INFORMÁTICA

2024

ÍNDICE GENERAL

CAPÍTULO

MONITORIZACIÓN DE SERVICIOS Y PROGRAMAS

3

3.1 Concepto de Monitor de Actividad

Definición 3.1.1 Carga

Tareas que ha de realizar el servidor, es decir, todo aquello que **demande recursos** del servidor.

Definición 3.1.2 Actividad

Operaciones que se realizan en el servidor como **consecuencia de la carga** que soporta

Definición 3.1.3 Monitor de actividad

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

¿Por qué monitorizar actividad?

- Saber qué hardware no es suficiente para nuestros propósitos y cuál sustituir (**cuello de botella**)
- Predecir situaciones (momentos de mayor carga, como por ejemplo black friday) [**capacity planning**].
- Conocer partes críticas de nuestro código (**hot spots**)
- **Adaptar el SO** a la carga.

3.1.1. Clasificaciones de monitores de actividad

Según cuánto se mide

- Monitor por **eventos**: Cada vez que ocurre algo (cambio en el sistema) da información exacta. Por ejemplo un archivo de log es un monitor por eventos.
- Monitor por **muestreo**: Cada T segundos (periodo de muestreo) el monitor mide. T puede variar. Otorga información estadística.
- Un monitor por **eventos no falla** pero si hay **muchos eventos perdemos el control** y en el monitor por **muestreo se controla la cantidad de información** que obtenemos.

Según cómo se mide

- **Software**: Programas instalados en el sistema. Cuando se ejecutan usan mi ram, mi cpu, mi almacenamiento, etc. La **sobrecarga es alta**.
- **Hardware**: Dispositivos físicos de medida. **No hacen sobrecarga**, muestran temperatura de partes del chasis, códigos de error de la placa, reloj de la cpu, etc.

Según interacción con el admin

- No interacción: La consulta sobre los resultados se realiza aparte mediante otra herramienta independiente al monitor (**segundo plano, batch monitors**).
- Sí interacción: Se pueden consultar los valores mientras monitorizas e interactuar con los datos (con gráficos, modificando parámetros) (**primer plano, on-line monitors**).

3.1.2. Atributos de un monitor

Definición 3.1.4 Anchura de entrada

Cuanta información **se almacena (recursos consumidos)** por mi monitor.

Definición 3.1.5 Sobrecarga

Qué recursos le "roba" el monitor al sistema.

Fórmula 3.1.1 Sobrecarga

Al usar mi monitor imagina que necesitaran 10 Bytes y tengo 100 Bytes de memoria, pues 10/100 = 10 % de mis recursos se gastan (sobrecarga) de mi servidor.

$$\text{Sobrecarga}_{\text{Recurso}}(\%) = \frac{\text{Recurso monitor}}{\text{Recurso sistema}} \cdot 100 \tag{3.1}$$

$$\text{Sobrecarga}_{\text{Recurso}}(\%) = \frac{\text{Recursos del monitor}}{\text{Capacidad total del recurso}} \cdot 100 \tag{3.2}$$

$$\text{Sobrecarga}_{\text{Recurso}}(\%) = \frac{\text{Tiempo de ejecución del monitor}}{\text{Intervalo de medida}} \cdot 100 \tag{3.3}$$

Un ejemplo: Sobrecarga de CPU, el monitor se activa cada 5 segundos, gasta 6ms de la CPU.

$\text{Sobrecarga}_{\text{CPU}}(\%) = \frac{6x10^{-3}s}{5s} \cdot 100 = 0,12\,\%$

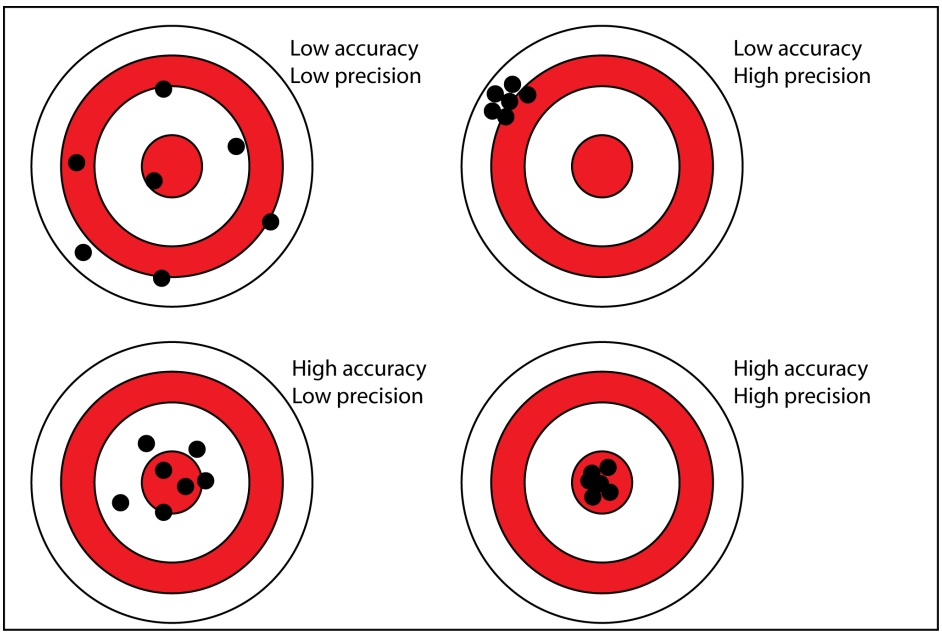

Figura 3.1: Precisión vs Exactitud (accuracy)

Definición 3.1.6 Exactitud
Lo que mido vs lo que debería estar midiendo de verdad. Mi reloj vs el reloj atómico del satélite. Si está descalibrado es un offset **(difrencia) entre el valor real y mi reloj**

Definición 3.1.7 Precisión
A veces se toma una medida varias veces y hay **diferencias entre ellas**.

Definición 3.1.8 Resolución
Cuánto tiene que **cambiar** el valor a medir **para detectar un cambio**

3.2 Monitorización a nivel de sistema

3.2.1. Directorio \proc

Es un **directorio virtual** utilizado por el núcleo de Linux para facilitar el **acceso** del usuario a las **estructuras de datos del S.O**. Es una forma encubierta de hacer una **llamada al sistema**. Hay

pseudoficheros como el tiempo que está la máquina encendida, hay carpetas con numeritos que son id de los procesos de la máquina y dentro de cada uno esa el estado del proceso, la memoria, el directorio actual, los ficheros abiertos, los hijos, etc. En proc hay **parámetros** que puedes cambiar para **modificar el funcionamiento** del sistema operativo.

3.2.2. uptime

Me dice el tiempo que lleva en marcha y la "carga media". Esto utiliza una constante (que determina la importancia de la carga actual) y la carga media previa. Pero, ¿qué es la carga del sistema?

Carga del sistema, según Linux

Hay 3 tipos de estados de un proceso:

1. **Ejecución (running + runnable)**: (o esperando a un core libre). Usa CPU.
2. **Bloqueado (blocked)**: Esperando acción de Entrada/Salida para continuar.
3. **Durmiendo (interruptible sleep)**: Esperando evento de usuario o similar.

> **Definición 3.2.1 Carga del sistema (Linux)**
> Número de procesos en **ejecución o bloqueados**.

3.2.3. ps (process status)

Muestra información sobre el **estado actual de los procesos** del sistema. Cuenta el 100 % de los recursos relativo a tus recursos en la máquina. Si tienes, por ejemplo, 4 CPUs, entonces 4 será 400 %. Ps tiene **mucho overhead** y **hay que llamarlo cada cierto tiempo**, así que no nos ayuda mucho en nuestro entorno.

3.2.4. top

Muestra **cada T segundos** valores globales de **carga**, **procesos**, utilización de **CPU**, uso de **DRAM y SWAP**, y valores particulares de cada proceso. Tiene **mucho overhead** porque es en **primer plano** y cada cierto tiempo me lo actualiza (también es **interactivo**).

Algunos valores importantes a saber

- US: Tiempo ejecutando procesos del usuario a los que no se les ha cambiado la prioridad.
- SY: Tiempo ejecutando procesos del kernel.
- ID: Idle, no hay nada que hacer.
- WA: No hay nada que hacer PORQUE estoy esperando entrada/salida y si fuera más rápida seguro estaría trabajando.

Procesos a mostrar:
-A, -e: show all processes; T: all processes on this terminal; U: processes for specified users…
Campos que mostrar: process ID, cumulative user time, number of minor/major page faults, parent process ID, RSS, time process was started, user ID number, user name, total VM size in bytes, kernel scheduling priority, etc.

```
$ 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
```

Figura 3.2: Comando ps y su salida.

```
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 us, 0.3 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem:   256464 total,  234008 used,  22456 free, 13784 buffers
KiB Swap:  136512 total,    4356 used, 132156 free,  5240 cached Mem

 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
```

Figura 3.3: Comando top y su salida.

3.2.5. vmstat (virtual memory statistics)

Mientras que top es puntual pero a nivel global de recursos y consume demasiado, vmstat resuelve esos problemas. Da **información cada cierto tiempo** sobre procesos que están **running**, **idle** o **blocked**. Tiene un fichero en proc y un **overhead mínimo**. La **primera línea** es **info desde el inicio** del sistema y **no nos sirve** para nada.

```
% 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
```

Figura 3.4: Comando vmstat y su salida.

Algunos elementos de vmstat

- procs (procesos):
  - r: running/runnable
  - b: Bloqueados por E/S
- io (e/s):
  - bi: blocks in
  - bo: blocks out
- swap (memoria virtual):
  - si: swapped in
  - so: swapped out
- system (sistema en general):
  - in: interrupciones por seg.
  - cs: cambios de contexto por seg.

3.2.6. sar (system activity reporter)

Recopila información sobre la **actividad del sistema actual** (interactivo) y **histórica** (en /var/log/sysstat/saDD donde DD indica día del mes, pero puedes cambiarlo). **Sar** es el **frontend**, mientras que **sadc** es el que a a proc y mide la información.

Monitorizar CPU

Ver 3.6 y 3.7.

Monitorizar unidades de almacenamiento

Ver 3.8.

```
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 global del procesador (opción por defecto)
-P      Mostrar estadísticas por cada núcleo (-P ALL: todos)
-I      Estadísticas sobre interrupciones
-w      Cambios de contexto
-q      Tamaño de la cola y carga media del sistema
-b      Estadísticas globales 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
...
```

Figura 3.5: Comando sar, en negrita lo que hay que saber para el examen.

Información de sar sobre discos

- **tps**: transferencias por segundo.
- **rd_sec/s y wr_sec/s**: lecturas por segundo y escrituras por segundo.
- **rq**: bloques que pide
- **await**: tiempo de respuesta del disco. Toma en cuenta tiempo de cola y servicio.
- **svctm**: tiempo de servicio una vez superada la cola.
- **util**: porcentaje de utilizacion. No es capacidad de almacenamiento, sino el ancho de banda del sata para utilizar el disco.
- **-b**: entrada/salida
- **-d**: discos

Monitorizar red

Ver 3.9.

Almacenamiento de datos del data collector (sadc)

Se utiliza un fichero de datos por día y se programa la ejecución de sadc x veces por cada fichero con cron. Además se añade un registro binario con los datos al histórico diario. Ver 3.10

> **Nota** **Calcular la anchura de entrada (carga) del monitor**

```
$ sar (=sar -u)
00:00:00 CPU  %usr  %nice %sys  %wa  %st   %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
00:15:00 ...
```

Figura 3.6: Por defecto me da la info. recopilada de hoy, (me da lo mismo que vmstat) a partir de las 0 h de ese día.

```
$ sar -P ALL 1
19:30:39 CPU  %usr  %nice %sys  %wa  %st   %idle
19:30:40 all  53.45  0.00 6.18  0.00 0.00  40.37
19:30:40   0  49.49  0.00 5.05  0.00 0.00  45.45
19:30:40   1  51.61  0.00 6.45  0.00 0.00  41.94
19:30:40   2  58.16  0.00 8.17  0.00 0.00  33.67
19:30:40   3  54.55  0.00 5.05  0.00 0.00  40.40
19:30:40 CPU  %usr  %nice %sys  %wa  %st   %idle
19:30:41 all  50.49  0.00 6.19  0.00 0.00  43.32
...
```

Figura 3.7: -P de cada CPU, si pongo 1 de la primera y ALL son todos. Lo desglosa por procesador y el global. Con el numerito indica el intervalo de muestreo (modo **interactivo**) y cada segundo mide la información.

> *Primero miramos el archivo de **/var/log/sysstat** con **ls**. Lo que más nos interesa es cuántos bytes tiene y cuándo fue la última modificación. Tenemos que pensar: Si la última muestra fue, por ejemplo, a las 23:58, ¿cada cuánto actualiza sar? Pues 23:58 es 1438 minutos, ¿por qué numero es divisible eso? Por 2 y por 719 (y por 2x719). Pues, si asumimos que se ejecuta cada 2 minutos, solo tenemos que dividir el tiempo del día entre 2 para saber cuántas muestras tomaríamos. Por nuestra descomposición de antes sabemos que 1438/2 es 719, así que esas son las muestras que tomamos. ¿Cuál es la carga? **Peso del archivo/num muestras**. Si pesara 3MB, pues 3MB/719 muestras. Si solo tomamos una muestra, pues la carga serían 3/1 = 3 MB.*

3.2.7. Método USE (Utilization, Saturation, Errors)

Para cada recurso comprobamos, según un intervalo:

1. **Utilización media**: tiempo de CPU (con sar -P, por ejemplo), memoria ocupada (vmstat), ancho de banda del disco...

```
$ sar -b -s 10:00:00 -e 12:00:00 -f /var/log/sysstat/sa06
10:00:00    tps     rtps  wtps  bread/s  bwrtn/s
10:05:00    0.74    0.39  0.35    7.96     3.27
10:10:00   65.12   59.96  5.16  631.62   162.64
10:15:00    ...
```

```
$ sar -d 10 2
18:46:09 DEV  tps   rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz  await  svctm  %util
18:46:19 sda  1.70   33.60      0.00     19.76      0.00     0.47   0.47   0.08
18:46:19 sr0  0.00    0.00      0.00      0.00      0.00     0.00   0.00   0.00
18:46:19 DEV  tps   rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz  await  svctm  %util
18:46:29 sda  8.60  114.40    518.10     73.55      0.06     7.12   0.93   0.80
18:46:29 sr0  0.00    0.00      0.00      0.00      0.00     0.00   0.00   0.00
```

Figura 3.8: Ejemplo, pido el día 6 del mes mas allá de las 10, el de abajo pido cada 10 segundos 2 muestras.

```
$ sar -n TCP,ETCP,DEV 1
Linux 3.2.55 (test-e4f1a80b)     08/18/2014     _x86_64_  (8 CPU)

09:10:43 PM  IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s rxcmp/s txcmp/s  rxmcst/s
09:10:44 PM     lo     14.00     14.00      1.34      1.34    0.00    0.00      0.00
09:10:44 PM   eth0   4114.00   4186.00   4537.46  28513.24    0.00    0.00      0.00

09:10:43 PM  active/s passive/s   iseg/s    oseg/s
09:10:44 PM     21.00      4.00   4107.00  22511.00

09:10:43 PM  atmptf/s  estres/s retrans/s isegerr/s   orsts/s
09:10:44 PM      0.00      0.00     36.00      0.00      1.00
[…]
```

Figura 3.9: -n es la conexión de red. Para este monitor la e/s la desglosa en unidades de almacenamiento y red. En otros casos podríamos juntarlo pero para este monitor no. En este ejemplo, se ve cada segundo TCP y sus errores

2. **Saturación**: Ocupación media de las colas (por ejemplo, sar -d con avgqu-sz)
3. **Errores**: Mensajes de error de los recursos (por ejemplo, dmesg)

3.2.8. Herramientas de monitorización de servidores distribuidos

Normalmente si tienes varios servidores controlados desde el ordenador del administrador, buscas programas que tengan **interfaz gráfica** como Elastic, Zabbix o Grafana.

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

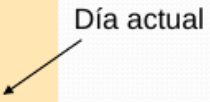

Figura 3.10: En este ejemplo guarda información cada 5 minutos (por eso las ediciones de los dias anteriores acaban a las 23:55) y el día actual del mes es 8 porque aún no escribió el último registro (el último es de las 18:45)

Características

- **Escalables**: Puedes tener **mini monitores** encargados de subconjuntos de servers. También puedes tener **servidores independientes** para base de datos y front end y **descubrir automáticamente** nuevos servers.
- **Extensibilidad**: Plugins para más funcionalidad (**scripts**), añades otros protocolos (como SNMP), añades API para apliicaciones (como Grafana).
- **Prestaciones**: permitir *polling* (buscar constantemente información) o *pushing* (mandar info automáticamente) con o sin agente. Envías por lotes la info o la comprimes para no quedarte sin espacio.
- **Fiabilidad**: Validar valores que recibes.
  - **+ disponibiliad**: te mandan alarmas y notificaciones.
- **Seguridad**: Encriptar comunicación y evitar suplantación con hashing.

3.3 Monitorización a nivel de aplicación (profiling)

Definición 3.3.1 Profiler

Sirve para saber qué partes de mi programa se usan y cuánto para poder optimizar el código.

Definición 3.3.2 Hot spot

Parte del código donde está la mayoría del tiempo de ejecución.

3.3.1. /usr/bin/time

Mide el tiempo de ejecución de un programa y muestra algunas estadísticas sobre su ejecución. (**NO ES UN PROFILER AÚN**) Es **diferente a time**, hay que poner especifícamente el directorio.

Estadísiticas

- **User time**: tiempo de CPU ejecutando en modo usuario.
- **System time**: tiempo de CPU ejecutando código del kernel.
- **Elapsed (wall clock) time**: tiempo que ha pasado desde el comienzo hasta el fin del programa.
- **Major page faults**: fallos de página que requieren acceder al almacenamiento permanente.
- **Cambios de contexto voluntarios**: Cuando acaba el programa o al tener que esperar a una operación de E/S cede la CPU a otro proceso.
- **Cambios involuntarios**: Expira su "*time slice*" (su tiempo asignado de cpu o e/s).

3.3.2. gprof

Mide tiempo de cpu de una hebra de un programa en c o c++ y está incluido en gcc (cuando compile mi programa añado pg).

Funcionamiento

1. **Arranque**:
   1. Genera **tabla con dirección en memoria** de cada función.
   2. **Contadores** de cada función de programa a 0 (**veces que se ejecuta** y estimar **tiempo en CPU**)
   3. El SO pone un **temporizador**.
2. **Ejecución**:
   1. Cada vez que **ejecuta una función**: Subir numero de ejecuciones y mira qué funcion la ha llamado.
   2. Cada vez que el **temporizador llega a 0seg**, se interrumpe el programa y se suma el contador de función interrumpida. Reinicia el contador.
3. **Final**:
   1. Teniendo en cuenta el tiempo total de CPU y los contadores de CPU, estima el CPU de cada función.
   2. Genera *flat profile* (cuál función gasta más tiempo) y *call profile* (cuál función gasta más tiempo de llamadas de sistema).

flat profile

- **% time**: % del tiempo total de CPU del programa que usa código propio de la función (no las llamadas a otras funciones)
- **Cumulative seconds**: La suma acumulada de los segundos consumidos (CPU) por el código propio de la función y por el de las funciones que aparecen encima de ella en la tabla.
- **Self seconds**: tiempo (CPU) total de ejecución del código propio de la función.
- **Self s/call**: tiempo (CPU) medio de ejecución del código PROPIO por cada llamada a la función.
- **Total s/call**: tiempo (CPU) medio de ejecución de cada llamada a la función (propio + llamadas).

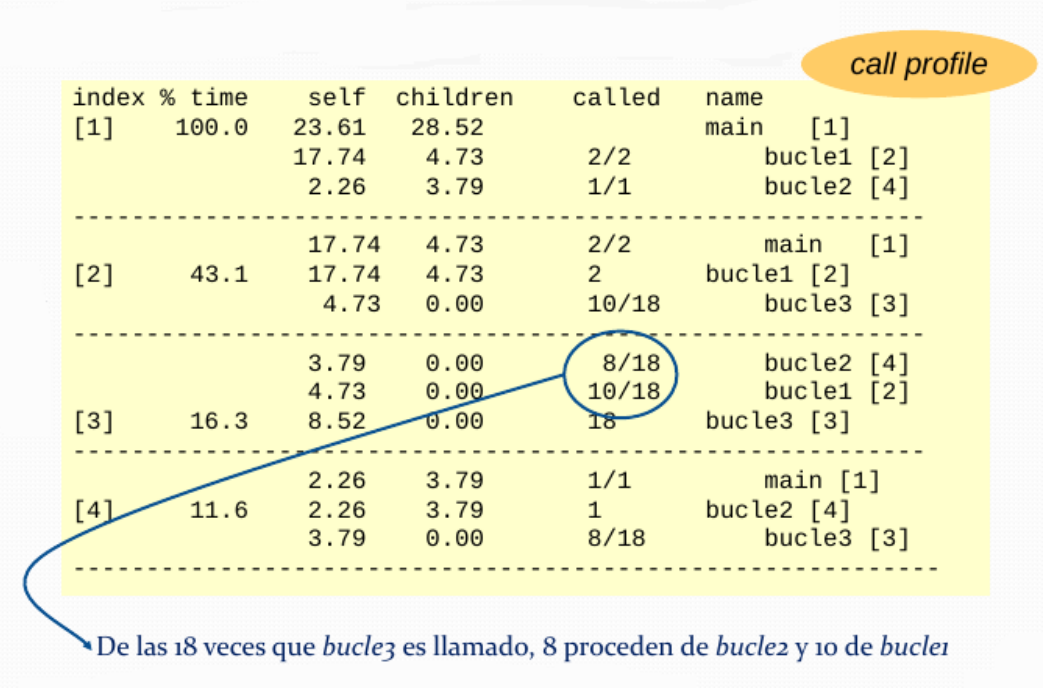

Figura 3.11: Explicación de call profile con un ejemplo.

call profile

Ver 3.11.

3.3.3. Perf

Es un conjunto de herramientas para el análisis de rendimiento que usa contadores hardware presentes en los microprocesadores (por ejemplo uenta cuánto se abre un archivo). Permiten analizar un hilo, un proceso + sus hijos, todos los procesos que se ejecutan en una CPU o todos los procesos que se ejecutan en el sistema.

Algunos comandos

- **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**.

3.3.4. Valgrind

Traduce a lenguaje propio el programa y simula lo que ocurriría. Es un ejecutable que tarda mucho más que el ejecutable original (coste computacional alto). En vez de una estimación, este va a decir cuantos fallos de página, caché... Emula la ejecución del programa.

3.3.5. V-Tune (Intel) y CodeXL (AMD)

Tanto AMD como intel desarrollan sus propios profilers. Funcionan como perf, pueden funcionar standalone (independiente del programa) y remotamente (ejecutado desde otro ordenador).