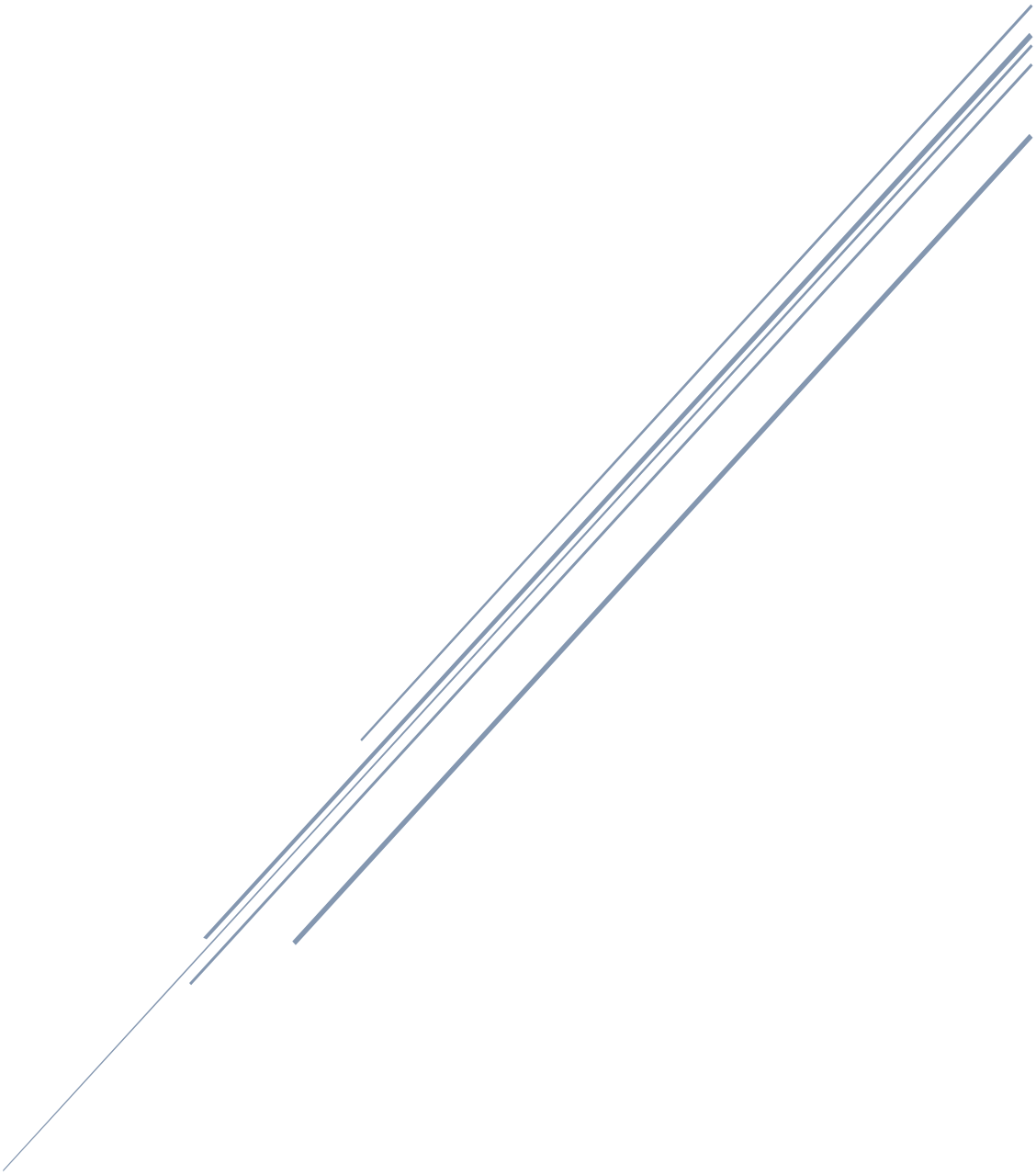
EVALUACIÓN DE CONFIGURACIONES

Práctica 3

–

Perfilado con ejecutables



FDI - UCM

Iván Aguilera Calle – Daniel García Moreno

1. Perf

En este apartado utilizaremos la herramienta perf para realizar análisis y obtener distintos datos.

Comenzamos consultando algunas páginas del manual:

- **perf.**
- **perf-list:** si lo ejecutamos, nos muestra un listado de los eventos disponibles.
- **perf-stat** nos muestra distintas estadísticas.
- **perf-record** guarda una muestra de una aplicación determinada en el archivo perf.data:
- **perf-report** sirve para visualizar el contenido del archivo generado con el anterior comando.

1.1. Cuenta de eventos

En este apartado obtendremos estadísticas de rendimiento para el programa "matrix1.c", usando para ello la orden perf stat -r 5 (realiza cinco ejecuciones).

```
ivan@ivan-System-Product-Name:~/Escritorio/P3$ perf stat -r 5 ./matrix1

 Performance counter stats for './matrix1' (5 runs):

 677,138501      task-clock (msec)                 #    1,000 CPUs utilized ( +-  0,39% )
             1      context-switches                 #    0,002 K/sec ( +- 17,50% )
             0      cpu-migrations                   #    0,000 K/sec
           619      page-faults                      #    0,914 K/sec ( +-  0,12% )
   2.630.671.673      cycles                           #    3,885 GHz   ( +-  0,16% )
  <not supported>      stalled-cycles-frontend
  <not supported>      stalled-cycles-backend
   4.552.352.311      instructions                     # 1,73  insns per cycle ( +-  0,00% )
     217.923.761      branches                         # 321,830 M/sec ( +-  0,00% )
         367.308      branch-misses                    # 0,17% of all branches ( +-  0,06% )

    0,677341930 seconds time elapsed                 ( +-  0,39% )
```

En el comando anteriormente ejecutado se puede observar distintos datos como es el número de cambios de contexto, fallos de página…

De todas las métricas, el cambio de contexto es la que tiene una mayor desviación (17.50%), por lo que, en este ejemplo, es la métrica menos precisa.

También observaremos el número de accesos y fallos de la cache de datos de primer nivel del programa “matrix1.c”, monitorizando para ello los siguientes eventos:

- L1-dcache-loads
- L1-dcache-load-misses
- L1-dcache-stores
- L1-dcache-store-misses

Ejecutando los comandos anteriores para el programa “matrix1.c”, obtenemos los siguientes resultados:

```
ivan@ivan-System-Product-Name:~/Escritorio/P3$ perf stat -e L1-dcache-loads,L1-dcache-load-
misses, L1-dcache-stores,L1-dcache-store-misses ./matrix1

 Performance counter stats for './matrix1':

    1.949.746.087      L1-dcache-loads
      266.319.197      L1-dcache-load-misses     #   13,66% of all L1-dcache hits
      434.957.370      L1-dcache-stores
   <not supported>     L1-dcache-store-misses

      0,698348513 seconds time elapsed
```

Repetimos los comandos anteriores, pero esta vez para el programa “matrix2.c” (similar al anterior pero que intercambia los dos bucles internos) y obtenemos los siguientes resultados:

```
ivan@ivan-System-Product-Name:~/Escritorio/P3$ gcc -g -o matrix2 matrix2.c

ivan@ivan-System-Product-Name:~/Escritorio/P3$ perf stat -e L1-dcache-loads,L1-dcache-load-
misses,L1-dcache-stores,L1-dcache-store-misses ./matrix2

 Performance counter stats for './matrix2':

    2.814.091.166      L1-dcache-loads
      27.390.859       L1-dcache-load-misses     #    0,97% of all L1-dcache hits
      434.946.178      L1-dcache-stores
   <not supported>     L1-dcache-store-misses

      0,608834616 seconds time elapsed
```

Comparando los dos resultados anteriores podemos ver que el programa matrix2 tiene un número más elevado de cache-loads.

En cuanto al número de fallos de cache-loads, matrix1 tiene un mayor número de cache-load fallos (13,66% en caso de matrix1 y 0.97% de matrix2), por lo que esto puede ser una causa de que el programa matrix1 tarde más en ejecutarse.

En cuanto al tiempo de ejecución el programa matrix1 tarda 0.69s y el programa matrix2 tarda 0.60s.

Como observación se puede apreciar que el evento L1-dcache-store-misses no está soportado.

1.2. Perfilado basado en eventos – 1

En este apartado obtendremos el tiempo consumido por cada función del programa "edges.c". Para ello, hemos realizado el muestreo utilizando la orden perf record y hemos generado el informe ejecutando perf report --stdio.

```
ivan@ivan-System-Product-Name:~/Escritorio/P3$ gcc -g -o edges edges.c
ivan@ivan-System-Product-Name:~/Escritorio/P3$ perf record ./edges img.pgm out.pgm


ivan@ivan-System-Product-Name:~/Escritorio/P3$ perf report --stdio
Samples in kernel modules can't be resolved as well.

# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 12K of event 'cycles:pp'
# Event count (approx.): 11806578382
#
# Overhead  Command  Shared Object     Symbol
# ........  .......  .................  ......................
#
    75.95%  edges    edges             [.] gaussian
    21.68%  edges    edges             [.] laplacian
     0.80%  edges    libc-2.23.so      [.] _IO_getc
     0.77%  edges    libc-2.23.so      [.] fputc
     0.28%  edges    edges             [.] save_image_file
     0.17%  edges    edges             [.] load_image_file
     0.09%  edges    edges             [.] fputc@plt
     0.04%  edges    edges             [.] fgetc@plt
     0.03%  edges    [unknown]         [k] 0xffffffff810c3b00
     0.03%  edges    [unknown]         [k] 0xffffffff81402957
```

Como ocurría en la práctica anterior, las funciones que mayor sobrecarga implican son "gaussian" y "laplacian" (75.95% y 21.68% respectivamente).

1.3. Perfilado basado en eventos – 2

En este segundo apartado, repetiremos lo que hemos hecho en el apartado anterior probando distintas opciones del comando perf record:

- perf record -g: muestra un grafo de llamadas

```
ivan@ivan-System-Product-Name:~/Escritorio/P3$ perf record -g ./edges img.pgm out.pgm
[ perf record: Woken up 5 times to write data ]
[ perf record: Captured and wrote 1.115 MB perf.data (12166 samples) ]

ivan@ivan-System-Product-Name:~/Escritorio/P3$ perf report --stdio
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 12K of event 'cycles:pp'
# Event count (approx.): 11810204399
#
# Children      Self  Command  Shared Object     Symbol
# ........  ........  .......  .................  ......................
#
    99.97%     0.00%  edges    edges             [.] main
            |
            ---main
               |
               |--97.65%-- edges
               |          |
               |          |--75.98%-- gaussian
               |          |          |
               |          |          |--0.04%-- 0xffffffff818381b8
               |          |          |          0xffffffff8106b772
                                                                  …
```

En el grafo (parcial) anterior, se puede apreciar que el 97.65% del tiempo se dedica a la llamada a “edges”.

- perf record -e (muestrea con otros eventos interesantes como fallos de página, fallos de lectura o escritura de la cache de datos de primer nivel). En la columna del overhead se nos muestra el porcentaje de muestras tomadas cuando se ha ejecutado perf (en el 69.50% de las muestras se estaba ejecutando load_image_file). El número de samples nos indica el número de muestras que realmente hemos tomado, mientras que el valor de event count es el número total de eventos que han ocurrido durante el muestreo.

```
ivan@...:~/Escritorio/P3$ perf record -e page-faults,L1-dcache-load-misses ./edges img.pgm out.pgm
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.122 MB perf.data (2638 samples) ]

ivan@ivan-System-Product-Name:~/Escritorio/P3$ perf report --stdio

# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 8  of event 'page-faults'
# Event count (approx.): 1059
#
# Overhead  Command  Shared Object     Symbol
# ........  .......  .................  ......................
#
    69.50%  edges    edges            [.] load_image_file
    27.10%  edges    libc-2.23.so       [.] _init
     2.55%  edges    ld-2.23.so       [.] dl_main
     0.47%  edges    [unknown]          [k] 0xffffffff8132a79a
     0.19%  edges    [unknown]          [k] 0xffffffff814045e5
     0.09%  edges    libc-2.23.so       [.] _exit
     0.09%  edges    [unknown]          [k] 0xffffffff81402ce5


# Samples: 2K of event 'L1-dcache-load-misses'
# Event count (approx.): 2113526
#
# Overhead  Command  Shared Object     Symbol
# ........  .......  .................  ......................
#
    34.75%  edges    edges            [.] gaussian
    13.00%  edges    [unknown]          [k] 0xffffffff81402957
    11.05%  edges    edges            [.] laplacian
     7.43%  edges    [unknown]          [k] 0xffffffff81402ce5
     4.56%  edges    libc-2.23.so       [.] fputc
     3.45%  edges    libc-2.23.so       [.] _IO_getc
     2.22%  edges    [unknown]          [k] 0xffffffff811c10de
     2.22%  edges    [unknown]          [k] 0xffffffff81404806
     1.64%  edges    [unknown]          [k] 0xffffffff81244802
     1.32%  edges    [unknown]          [k] 0xffffffff81835e56
     0.77%  edges    [unknown]          [k] 0xffffffff811c312e
     0.67%  edges    [unknown]          [k] 0xffffffff8117bdec
     0.67%  edges    [unknown]          [k] 0xffffffff81835da4
     0.48%  edges    [unknown]          [k] 0xffffffff812f56f7
     0.42%  edges    [unknown]          [k] 0xffffffff81190ad7
     0.36%  edges    [unknown]          [k] 0xffffffff813fae12
     0.30%  edges    [unknown]          [k] 0xffffffff812447ad
     0.26%  edges    [unknown]          [k] 0xffffffff81245461
```

- perf record -F (distintas frecuencias de muestreo) -c (cuentas de eventos).

  Como el número de eventos en un programa debe ser similar entre distintas ejecuciones, al triplicar el valor de F (de 5 a 15) aumenta el número de muestras tomadas (el triple aproximadamente), por lo que a con una F de 15 los resultados serán más precisos.

```
daniel@daniel-Surface-Pro-2:~/Escritorio/P3$ sudo perf record -F 5 ./edges img.pgm out.pgm
daniel@daniel-Surface-Pro-2:~/Escritorio/P3$ sudo perf report --stdio
# To display the perf.data header info, please use --header/--header-only option
#
#
# Total Lost Samples: 0
#
# Samples: 11  of event 'cycles:ppp'
# Event count (approx.): 243992836343
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................  ......................
#
    98.69%  edges    libc-2.23.so       [.] _IO_getc
     0.64%  edges    edges              [.] laplacian
     0.43%  edges    edges              [.] gaussian
     0.21%  edges    edges              [.] save_image_file
     0.03%  edges    [kernel.kallsyms]  [k] flush_signal_handlers
     0.00%  perf     [kernel.kallsyms]  [k] native_write_msr

daniel@daniel-Surface-Pro-2:~/Escritorio/P3$ sudo perf record -F 15 ./edges img.pgm out.pgm
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.016 MB perf.data (38 samples) ]
daniel@daniel-Surface-Pro-2:~/Escritorio/P3$ sudo perf report --stdio# To display the perf.data header
info, please use --header/--header-only option
#
#
# Total Lost Samples: 0
#
# Samples: 38  of event 'cycles:ppp'
# Event count (approx.): 45539628551
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................  ......................
#
    87.51%  edges    libc-2.23.so       [.] _IO_getc
     7.96%  edges    edges              [.] gaussian
     4.15%  edges    edges              [.] laplacian
     0.36%  edges    libc-2.23.so       [.] fputc
     0.02%  perf     [kernel.kallsyms]  [k] native_sched_clock
     0.00%  perf     [kernel.kallsyms]  [k] native_write_msr
```

```
daniel@daniel-Surface-Pro-2:~/Escritorio/P3$ sudo perf record -c 5 ./edges img.pgm out.pgm
[ perf record: Woken up 467 times to write data ]
Warning:
Processed 3838692 events and lost 1 chunks!
[ perf record: Captured and wrote 117.178 MB perf.data (3835993 samples) ]
daniel@daniel-Surface-Pro-2:~/Escritorio/P3$ sudo perf report --stdio
# To display the perf.data header info, please use --header/--header-only options.
# Total Lost Samples: 0
#
# Samples: 3M of event 'cycles:ppp'
# Event count (approx.): 19179965
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................  .....................................
#
    24.01%  perf     [kernel.kallsyms]  [k] native_sched_clock
    10.48%  perf     [kernel.kallsyms]  [k] nmi_handle
     6.86%  perf     [kernel.kallsyms]  [k] sched_clock
     6.14%  perf     [kernel.kallsyms]  [k] do_nmi
     5.98%  perf     [kernel.kallsyms]  [k] intel_pmu_handle_irq
     5.17%  perf     [kernel.kallsyms]  [k] nmi_restore
…


daniel@daniel-Surface-Pro-2:~/Escritorio/P3$ sudo perf record -c 15 ./edges img.pgm out.pgm
[ perf record: Woken up 480 times to write data ]
[ perf record: Captured and wrote 119.895 MB perf.data (3924901 samples) ]
daniel@daniel-Surface-Pro-2:~/Escritorio/P3$ sudo perf report --stdio
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 3M of event 'cycles:ppp'
# Event count (approx.): 58873515
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................  .....................................
#
    23.98%  edges    [kernel.kallsyms]  [k] native_sched_clock
    10.47%  edges    [kernel.kallsyms]  [k] nmi_handle
     6.85%  edges    [kernel.kallsyms]  [k] sched_clock
     6.14%  edges    [kernel.kallsyms]  [k] do_nmi
     5.98%  edges    [kernel.kallsyms]  [k] intel_pmu_handle_irq
     5.17%  edges    [kernel.kallsyms]  [k] nmi_restore
     …
```

Ahora realizaremos un análisis comparativo entre las herramientas perf, gprof y google-pprof analizando el ámbito, precisión y sobrecarga: el gprof y el Google-pprof nos muestran más tiempos consumidos por cada función, en cambio el perf nos muestra el número de muestras mientras se estaba ejecutando una función (gaussian, laplacian…).

El perf tiene un ámbito más orientado a eventos mientras que el gprof está más orientados a conocer el tiempo consumido por cada función.

El Google-pprof nos dibuja de una manera más grafica el grafo de llamadas.

En cuanto a la sobrecarga, con el google-pprof, se obtuvo un tiempo sin realizar muestreo de 4.37s. Habría que obtener el tiempo de ejecución del programa edges con el perf stat y ver la sobrecarga que introduce.

2. Valgrind

En esta segunda parte, utilizaremos la herramienta cachegrind para obtener medidas del uso de la cache de primer nivel de los programas “matrix1.c” y “matrix2.c”, ejecutando para ello los siguientes comandos:

- Valgrind --tool=cachegrind ./matrix1

```
ivan@ivan-System-Product-Name:~/Escritorio/P3$ valgrind --tool=cachegrind ./matrix1
==6461== Cachegrind, a cache and branch-prediction profiler
==6461== Copyright (C) 2002-2015, and GNU GPL'd, by Nicholas Nethercote et al.
==6461== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==6461== Command: ./matrix1
==6461==
--6461-- warning: L3 cache found, using its data for the LL simulation.
==6461==
==6461== I   refs:      4,549,780,330
==6461== I1  misses:            757
==6461== LLi misses:            749
==6461== I1  miss rate:        0.00%
==6461== LLi miss rate:        0.00%
==6461==
==6461== D   refs:      2,166,879,428  (1,949,067,582 rd   + 217,811,846 wr)
==6461== D1  misses:      243,811,141  (  243,360,648 rd   +     450,493 wr)
==6461== LLd misses:          169,113  (       33,644 rd   +     135,469 wr)
==6461== D1  miss rate:        11.3% (        12.5%     +         0.2%  )
==6461== LLd miss rate:         0.0% (         0.0%     +         0.1%  )
==6461==
==6461== LL refs:         243,811,898  (  243,361,405 rd   +     450,493 wr)
==6461== LL misses:           169,862  (       34,393 rd   +     135,469 wr)
==6461== LL miss rate:          0.0% (         0.0%     +         0.1%  )
```

- Valgrind --tool=cachegrind ./matrix2

```
ivan@ivan-System-Product-Name:~/Escritorio/P3$ valgrind --tool=cachegrind ./matrix2
==6515== Cachegrind, a cache and branch-prediction profiler
==6515== Copyright (C) 2002-2015, and GNU GPL'd, by Nicholas Nethercote et al.
==6515== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==6515== Command: ./matrix2
==6515==
--6515-- warning: L3 cache found, using its data for the LL simulation.
==6515==
==6515== I   refs:      7,142,144,534
==6515== I1  misses:            758
==6515== LLi misses:            750
==6515== I1  miss rate:        0.00%
==6515== LLi miss rate:        0.00%
==6515==
==6515== D   refs:      3,030,881,830  (2,813,429,383 rd   + 217,452,447 wr)
==6515== D1  misses:       27,227,341  (   27,091,847 rd   +     135,494 wr)
==6515== LLd misses:          250,871  (     115,401 rd   +     135,470 wr)
==6515== D1  miss rate:           0.9% (         1.0%     +         0.1%  )
==6515== LLd miss rate:           0.0% (         0.0%     +         0.1%  )
==6515==
==6515== LL refs:          27,228,099  (   27,092,605 rd   +     135,494 wr)
==6515== LL misses:           251,621  (     116,151 rd   +     135,470 wr)
==6515== LL miss rate:            0.0% (         0.0%     +         0.1%  )
```

Como se puede ver en los resultados anteriores, podemos apreciar información más precisa que utilizando el comando perf. El programa “matrix2” sigue teniendo un menor número de fallo de cache de primer nivel y valgrind nos da información de cuántos son de lectura y cuántos son de escritura.

Con el valgrind estamos haciendo un seguimiento más exhaustivo del uso de la memoria.

3. Strace

En esta tercera y última parte utilizaremos la orden strace para trazar las llamadas al sistema que realiza un programa.

Observamos los ficheros que abre la orden vmstat, trazando la llamada al sistema open (utilizando la opción -e open de strace).

```
ivan@ivan-System-Product-Name:~/Escritorio/P3$ strace -e open vmstat
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libprocps.so.4", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libsystemd.so.0", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/librt.so.1", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/liblzma.so.5", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libgcrypt.so.20", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libpcre.so.3", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libgpg-error.so.0", O_RDONLY|O_CLOEXEC) = 3
open("/proc/filesystems", O_RDONLY)     = 3
open("/sys/devices/system/cpu/online", O_RDONLY|O_CLOEXEC) = 3
open("/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
open("/usr/share/locale/locale.alias", O_RDONLY|O_CLOEXEC) = 3
open("/usr/share/locale/es/LC_MESSAGES/procps-ng.mo", O_RDONLY) = -1 ENOENT (No such file
or directory)
open("/usr/share/locale-langpack/es/LC_MESSAGES/procps-ng.mo", O_RDONLY) = 3
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
open("/usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache", O_RDONLY) = 3
 r  b   swpd  libre búfer caché   si   so    bi    bo   in   cs us sy id wa st
open("/proc/meminfo", O_RDONLY)         = 3
open("/proc/stat", O_RDONLY)            = 4
open("/proc/vmstat", O_RDONLY)          = 5
 4  0      0 4872396 168436 1669888    0    0    66    81   83  516  6  1 92  0  0
+++ exited with 0 +++
```

- ¿De dónde saca toda la información vmstat?
    - De /proc/vmstat, de /proc/stat principalmente además de /proc/meminfo y de otras rutas más específicas.
- ¿De dónde saca la información ps?
    - Al igual que el comando anterior ejecutamos strace -e open ps
    - Del /proc/PID/stat
    - Del /proc/PID/status

- ¿De dónde saca la información top?
    - Al igual que el comando anterior ejecutamos strace -e open top
    - Del /proc/PID/stat
    - Del /proc/PID/statm
    - Del /proc/loadavg
    - …
- ¿De dónde saca la información sar?
    - Al igual que el comando anterior ejecutamos strace -e open sar

Ahora, obtendremos un resumen estadístico (opción -c) de las llamadas al sistema realizadas por la orden "find > /usr &> /dev/null". Usaremos la opción -o de strace para escribir la salida en un fichero:

```
ivan@ivan-System-Product-Name:~/Escritorio/P3$ strace -c -o E4_2.txt find /usr &> /dev/null

% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 59.43    0.001689           0     63586           getdents
 19.46    0.000553           0    286065           fcntl
 10.77    0.000306           0    126946           close
  3.69    0.000105           0     32012           openat
  3.52    0.000100           0     63560           newfstatat
  2.18    0.000062           0     32025           fstat
  0.67    0.000019           0       100           brk
  0.28    0.000008           0      3709           write
  0.00    0.000000           0        10           read
  0.00    0.000000           0        14        1 open
  0.00    0.000000           0        24           mmap
  0.00    0.000000           0        14           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         2           rt_sigaction
  0.00    0.000000           0         1           rt_sigprocmask
  0.00    0.000000           0         3        2 ioctl
  0.00    0.000000           0         8        8 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           uname
  0.00    0.000000           0         1           fchdir
  0.00    0.000000           0         1           getrlimit
  0.00    0.000000           0         2        2 statfs
  0.00    0.000000           0         1           arch_prctl
  0.00    0.000000           0         1           futex
  0.00    0.000000           0         1           set_tid_address
  0.00    0.000000           0         1           set_robust_list
------ ----------- ----------- --------- --------- ----------------
100.00    0.002842              608090        13 total
```

- ¿Para qué sirven las 5 llamadas más usadas?
    - getdents: lee algunas estructuras "linux_dirents" del directorio donde está el archivo.
    - fcntl: manipula el descriptor de fichero.
    - close: sirve para cerrar ficheros abiertos.
    - openat: abre un fichero relativo a un descriptor de directorio. Opera de manera similar al open.
    - newfstatat: devuelve información sobre un archivo