

Laborator 07 - Profiling & Debugging

Materiale ajutătoare

- lab07-slides.pdf [<http://elf.cs.pub.ro/so/res/laboratoare/lab07-slides.pdf>]

Nice to Watch

- Google I/O 2010 - Measure in milliseconds: Meet Speed Tracer [<http://www.youtube.com/watch?v=73IyVBMf2uY>]
- MIT Lecture: Performance Engineering with Profiling Tools [<http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-172-performance-engineering-of-software-systems-fall-2010/video-lectures/lecture-5-performance-engineering-with-profiling-tools/>]

Latency Comparison Numbers

Operation	Time (ns)	Notes
L1 cache reference	0.5 ns	
Branch mispredict	5 ns	
L2 cache reference	7 ns	14x L1 cache
Mutex lock/unlock	25 ns	
Main memory reference	100 ns	20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy	3,000 ns	
Send 1K bytes over 1 Gbps network	10,000 ns	0.01 ms
Read 4K randomly from SSD*	150,000 ns	0.15 ms
Read 1 MB sequentially from memory	250,000 ns	0.25 ms
Round trip within same datacenter	500,000 ns	0.5 ms
Read 1 MB sequentially from SSD*	1,000,000 ns	1 ms, 4x memory
Disk seek	10,000,000 ns	10 ms, 20x datacenter roundtrip
Read 1 MB sequentially from disk	20,000,000 ns	20 ms, 80x memory, 20x SSD
Send packet Caracal - NY - Caracal	150,000,000 ns	150 ms

Credits:

- By Jeff Dean: <http://research.google.com/people/jeff/> [<http://research.google.com/people/jeff/>]
- Originally by Peter Norvig: <http://norvig.com/21-days.html#answers> [<http://norvig.com/21-days.html#answers>]

Link-uri către secțiuni utile

Linux

- [Profiler și tehnici de profiling](#)
- [Unelte de profiling](#)
- [Utilizare perf](#)
- [Unelte de debugging](#)

Profiler și tehnici de profiling

Un **profiler** este un utilitar de analiză a **performanței** care ajută programatorul să determine punctele critice – **bottleneck** – ale unui program. Acest lucru se realizează prin investigarea comportamentului programului, evaluarea consumului de memorie și relația dintre modulele acestuia.

Suportul pentru profilare este disponibil la nivel de:

- **bibliotecă C** (GNU libc), prin informații legate de timpul de viață al alocărilor de memorie,
- **compiler**, prin modificarea codului în tehnica de **instrumentare** se poate realiza ușor în procesul de compilare, compilerul fiind cel ce inserează secțiunile de cod necesare,
- **nucleu** al sistemului de operare, prin punerea la dispoziție de apeluri de sistem specifice,
- **hardware**, unele procesoare sunt dotate cu contoare de temporizare (Time Stamp Counter - TSC [http://en.wikipedia.org/wiki/Time_Stamp_Counter]) sau contoare de performanță care numără evenimente precum cicluri de procesor sau TLB miss-uri.

Două moduri prin care se realizează profiling-ul sunt prezentate în continuare:

Tehnica de instrumentare (instrumentation)

Profiler-ele bazate pe această tehnică necesită de obicei **modificări** în codul programului: se inserează secțiuni de cod la începutul și sfârșitul funcției ce se dorește analizată. De asemenea, se rețin și funcțiile apelate. Astfel, se poate estima timpul total al apelului în sine cât și al apelurilor de subfuncții.

Dezavantajul major al acestor profilare este legat de modificarea codului: în funcții de dimensiune scăzută și des apelate, acest overhead poate duce la o interpretare greșită a rezultatelor.

Tehnica de eșantionare (sampling)

Profiler-ele bazate pe sampling **nu fac schimbări** în codul programului, ci verifică periodic procesorul cu scopul de a determina ce funcție (instrucțiune) se execută la momentul respectiv. Apoi estimează frecvența și timpul de execuție al unei anumite funcții într-o perioadă de timp.

Unelte de profiling

În continuare sunt prezentate câteva unelte folosite în profiling.

perfcounters

Majoritatea procesoarelor moderne oferă registre speciale (**performance counters**) care contorizează diferite tipuri de evenimente hardware: instrucțiuni executate, cache-miss-uri, instrucțiuni de salt anticipate greșit, fără să afecteze performanța nucleului sau a aplicațiilor. Aceste registre pot declanșa întreruperi atunci când se acumulează un anumit număr de evenimente și astfel se pot folosi pentru analiza codului care rulează pe procesorul în cauză.

Subsistemul perfcounters:

- se găsește în nucleul Linux începând cu versiunea 2.6.31 [<http://lwn.net/Articles/339361/>]
(CONFIG_PERF_COUNTERS=y)
- este înlocuitorul lui **oprofile**
- oferă suport pentru:
 - evenimente hardware (instrucțiuni, accese cache, ciclii de magistrală).
 - evenimente software (page fault, cpu-clock, cpu migrations).

- tracepoints (e.g: sys_enter_open, sys_exit_open).

perf

Utilitarul perf este interfața subsistemului perfcounters cu utilizatorul. Oferă o linie de comandă asemănătoare cu git și nu necesită existența unui daemon.

Un tutorial despre perf găsiți aici [<https://perf.wiki.kernel.org/index.php/Tutorial>].

Utilizare perf

```
$ perf [--version] [--help] COMMAND [ARGS]
```

Cele mai folosite comenzi sunt:

- annotated - Citește perf.data și afișează codul cu adnotări
- list - Listează numele simbolice ale tuturor tipurilor de evenimente ce pot fi urmărite de perf
- lock - Analizează evenimentele de tip lock
- record - Rulează o comandă și salvează informațiile de profiling în fișierul perf.data
- report - Citește perf.data (creat de perf record) și afișează profilul
- sched - Utilitar pentru măsurarea proprietăților planificatorului (latențe)
- stat - Rulează o comandă și afișează statisticile înregistrate de subsistemul performance counters
- top - Generează și afișează informații în timp real despre încărcarea unui sistem

perf list

- man perf-list [<http://manpages.ubuntu.com/manpages/natty/man1/perf-list.1.html>]

Afișează numele simbolice ale tuturor tipurilor de evenimente ce pot fi urmărite de perf.

```
$ perf list
List of pre-defined events (to be used in -e):

cpu-cycles OR cycles          [Hardware event]
instructions                  [Hardware event]

cpu-clock                     [Software event]
page-faults OR faults        [Software event]

L1-dcache-loads              [Hardware cache event]
L1-dcache-load-misses        [Hardware cache event]

rNNN                         [Raw hardware event descriptor]

mem:<addr>[:access]           [Hardware breakpoint]

syscalls:sys_enter_accept     [Tracepoint event]
syscalls:sys_exit_accept      [Tracepoint event]
```

Atunci când un eveniment nu este disponibil în forma simbolică, poate fi folosit cu perf în forma procesorului din sistemul analizat.

perf stat

- perf-stat [<http://manpages.ubuntu.com/manpages/lucid/man1/perf-stat.1.html>]

Rulează o comandă și afișează statisticile înregistrate de subsistemul performance counters.

```
$ perf stat ls -R /usr/src/linux
Performance counter stats for 'ls -R /usr/src/linux':

      934.512846 task-clock-msecs      #      0.114 CPUs
        1695 context-switches        #      0.002 M/sec
         163 CPU-migrations           #      0.000 M/sec
          306 page-faults             #      0.000 M/sec
    725144010 cycles                  #    775.959 M/sec
   419392509 instructions             #      0.578 IPC
    80242637 branches                 #     85.866 M/sec
    5680112 branch-misses             #      7.079 %
   174667968 cache-references         #    186.908 M/sec
    4178882 cache-misses              #      4.472 M/sec

      8.199187316 seconds time elapsed
```

perf stat oferă posibilitatea colectării datelor în urma rulării de mai multe ori a unui program specificând opțiunea -r.

```
$ perf stat -r 6 sleep 1
Performance counter stats for 'sleep 1' (6 runs):

    1.757147 task-clock-msecs #      0.002 CPUs ( +- 3.000% )
          1 context-switches #      0.001 M/sec ( +- 14.286% )
           0 CPU-migrations  #      0.000 M/sec ( +- 100.000% )
         144 page-faults     #      0.082 M/sec ( +- 0.147% )
   1373254 cycles            #    781.525 M/sec ( +- 2.856% )
    588831 instructions       #      0.429 IPC ( +- 0.667% )
    106846 branches          #     60.806 M/sec ( +- 0.324% )
     11312 branch-misses     #     10.587 % ( +- 0.851% )
   1.002619407 seconds time elapsed ( +- 0.012% )
```

Observați mai sus evenimentele cele mai importante contorizate.

perf top

- man perf-top [<http://manpages.ubuntu.com/manpages/natty/man1/perf-top.1.html>]

Generează și afișează informații în timp real despre încărcarea unui sistem.

```
$ ls -R /home
$ perf top -p $(pidof ls)

-----
PerfTop:      181 irqs/sec kernel:72.4% (target_pid: 10421)
-----
  samples  pcnt function                                DSO
  -----  -
  270.00  15.8% __d_lookup                                     [kernel.kallsyms]
  145.00   8.5% __GI___strcoll_l                          /lib/libc-2.12.1.so
   99.00   5.8% link_path_walk                             [kernel.kallsyms]
   97.00   5.7% find_inode_fast                           [kernel.kallsyms]
   91.00   5.3% __GI_strncmp                               /lib/libc-2.12.1.so
   55.00   3.2% move_freepages_block                       [kernel.kallsyms]
   44.00   2.6% ext3_dx_find_entry                         [kernel.kallsyms]
   41.00   2.4% ext3_find_entry                           [kernel.kallsyms]
   40.00   2.3% dput                                         [kernel.kallsyms]
   39.00   2.3% ext3_check_dir_entry                       [kernel.kallsyms]
```

Observăm că funcțiile de lucru cu fișiere (parcursare, căutare) sunt cele care apar cel mai des în outputul lui perf-top corespunzător rulării comenzii de listare recursivă a directorului home.

perf record

- `man perf-record` [<http://manpages.ubuntu.com/manpages/natty/man1/perf-record.1.html>]

Rulează o comandă și salvează informațiile de profiling în fișierul `perf.data`.

```
$ perf record wget http://elf.cs.pub.ro/so/wiki/laboratoare/laborator-07

[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.008 MB perf.data (~334 samples) ]

$ ls
laborator-07  perf.data
```

perf report

- `man perf-report` [<http://manpages.ubuntu.com/manpages/natty/man1/perf-report.1.html>]

Interpretează datele salvate în `perf.data` în urma analizei folosind `perf record`. Astfel pentru exemplul `wget` de mai sus avem:

```
$ perf report
# Events: 13 cycles
#
# Overhead  Command      Shared Object  Symbol
# .....  ....
#
# 86.43%    wget          e8ee21         [.] 0x00000000e8ee21
# 11.03%    wget [kernel.kallsyms] [k] prep_new_page
# 2.37%    wget [kernel.kallsyms] [k] sock_aio_read
# 0.11%    wget [kernel.kallsyms] [k] perf_event_comm
# 0.05%    wget [kernel.kallsyms] [k] native_write_msr_safe
```

Unelte de debugging

strace

`strace` interceptează și înregistrează apelurile de sistem făcute de un proces și semnalele pe care acesta le primește. În cea mai simplă formă `strace` rulează comanda specificată până când procesul asociat se încheie.

```
$strace cat /proc/cpuinfo
execve("/bin/cat", ["cat", "/proc/cpuinfo"], [/* 30 vars */]) = 0
open("/proc/cpuinfo", O_RDONLY) = 3
read(3, "processor\t: 0\nvendor_id\t: Genuin...", 32768) = 3652
write(1, "processor\t: 0\nvendor_id\t: Genui...", 7512) = 7512
```

Cele mai folosite opțiuni pentru `strace` sunt:

- `-f`, cu această opțiune vor fi urmărite și procesele copil create de procesul curent
- `-o filename`, în mod implicit `strace` afișează informațiile la `stderr`. Cu această opțiune, output-ul va fi pus în fișierul `filename`
- `-p pid`, `pid`-ul procesului de urmărit.
- `-e expresie`, modifică apelurile urmărite.

- ```
daniel@debian$ strace -f -e connect,socket,bind -p $(pidof iceweasel)
Process 6429 attached with 30 threads - interrupt to quit
socket(PF_INET, SOCK_STREAM, IPPROTO_IP) = 50
```

```
connect(50, {sa_family=AF_INET, sin_port=htons(80), sin_addr=inet_addr("141.85.227.65")}, 16) = -1 EINPROGRESS
```

Un alt utilitar înrudit cu `strace` este `ltrace` [<http://linux.die.net/man/1/ltrace>]. Acesta urmărește apelurile de bibliotecă.

## `gdb`

Scopul unui debugger (de exemplu GDB, prin comanda `gdb`) este să ne permită să inspectăm ce se întâmplă *în interiorul* unui program în timp ce acesta rulează sau la terminarea execuției acestuia, în momentul când s-a produs o eroare fatală, pe baza fișierului *coredump* creat de către acesta.

Mai multe detalii în secțiunea de resurse [<http://ocw.cs.pub.ro/courses/so/laboratoare/resurse/gdb>].

## `valgrind`

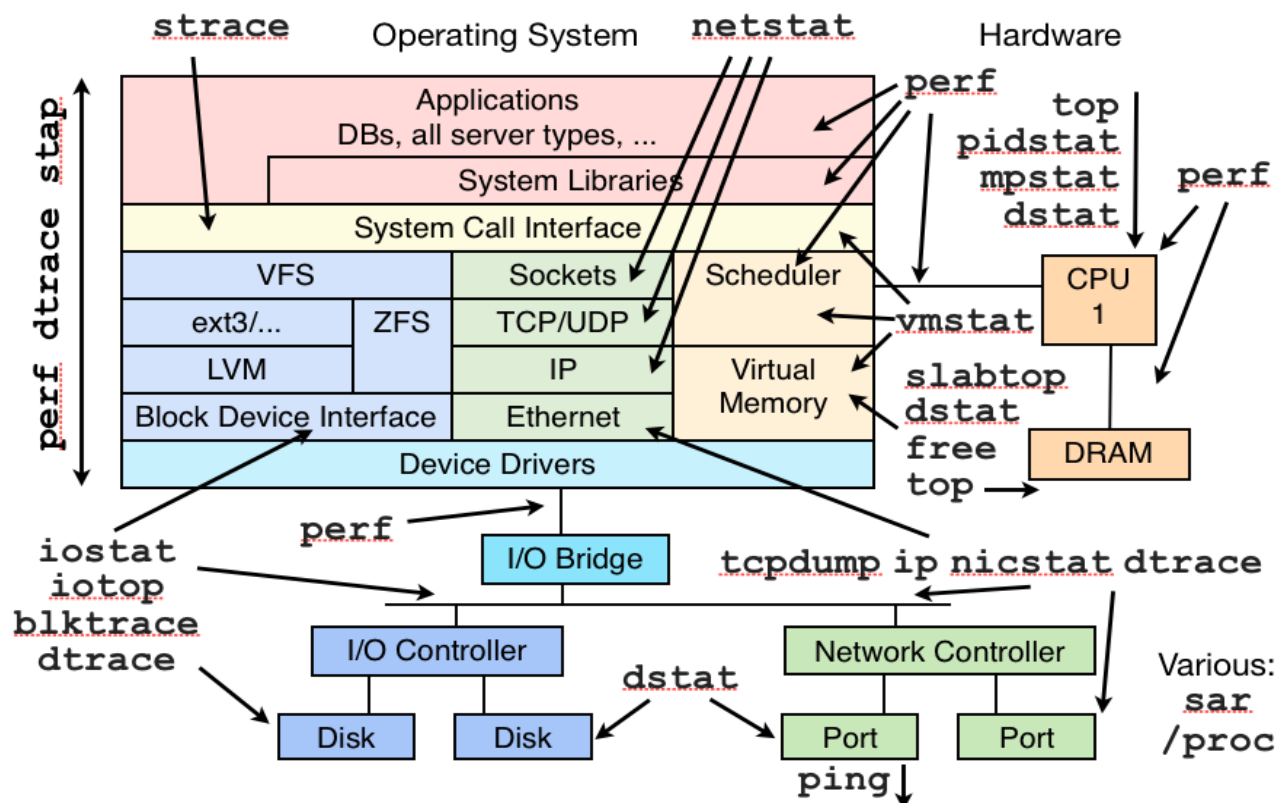
Valgrind reprezintă o suită de utilitare folosite pentru debugging și profiling. Printre cele mai folosite tool-uri sau unelte din această suită sunt:

- `memcheck` - Realizează analiza utilizării memoriei
- `cachegrind` - Realizează analiza performanței cache-ului
- `helgrind` - Folosit pentru depanarea condițiilor de cursă în cadrul programelor cu multiple fire de execuție (multithreaded).

Mai multe detalii aici [<http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-05#valgrind>].

## Alte utilitare

# Analysis and Tools



- Oprofile [<http://oprofile.sourceforge.net/download>]
- Kernrate [<http://www.microsoft.com/downloads/details.aspx?familyid=d6e95259-8d9d-4c22-89c4-fad382eddc1&displaylang=en>] este un echivalent al oprofile pentru Windows.
- KCachegrind [<http://kcache.grind.sourceforge.net/html/Home.html>]
- perf-tools [<http://code.google.com/p/google-perftools/>]
- XPerf [<http://metadataconsulting.blogspot.com/2017/05/How-to-install-and-use-Windows-Performance-Toolkit.html>]
- GNU gprof [<http://sourceware.org/binutils/docs/gprof>]

## Exerciții

Folosiți arhiva lab07-tasks.zip [<http://elf.cs.pub.ro/so/res/laboratoare/lab07-tasks.zip>] aferentă laboratorului.

### Exercițiul 0 - Joc interactiv

- Detalii desfășurare joc [[http://ocw.cs.pub.ro/courses/so/meta/notare#joc\\_interactiv](http://ocw.cs.pub.ro/courses/so/meta/notare#joc_interactiv)].

## Linux

Întrucât avem nevoie de suport hardware, suport inexistent pe mașina virtuală, **lucrați pe sistemul fizic**.

Pentru a vedea ce pachet trebuie să instalați, rulați comanda `perf` fără parametri.

Pentru a putea face exercițiile e nevoie de utilitarul `linux-tools`. Puteți verifica asta rulând comanda `perf --help`. Dacă comanda nu e găsită, trebuie să instalați pachetul:

```
student@so:~$ sudo apt-get update
student@so:~$ sudo apt-get install linux-tools-4.15.0-34-generic
```

## Exercițiul 1 - Custom Profiling

Scopul exercițiului este să analizăm numărul de instrucțiuni executate de către procesor pe baza unui executabil. Perf pune la dispoziție un mod de a extrage datele importante din profiling prin suportul de scripting oferit de `perf script`. Acesta funcționează împreună cu `perf record` care obține lista de sample-uri și o salvează în fișierul `perf.data`.

Intrați în directorul `1-custom`. Primul pas este să generăm fișierul `perf.data` care conține sampleurile. Pentru asta executați :

```
make
sudo perf record -e cycles:pp -c 10000 -d ./hash
```

Folosiți comanda `perf script` cu opțiunea `-F` (investigați `man perf-script`) astfel încât să găsiți numărul total de valori ale `instruction pointer`-ului și apoi pe cele aflate în funcția `hash_search_index`. Având cele două valori, calculați procentul valorilor din funcția `hash_search_index`.

Folosiți `wc -l` pentru a număra liniile outputului și `grep` pentru a filtra după simbolul `hash_search_index`. Pentru a face calcule cu numere raționale folosiți o comandă de tipul: `echo 7/2 | bc -l`.

Opțional, puteți folosi script-ul pus la dispoziție în cadrul laboratorului pentru a calcula numărul de accese în cadrul funcției `hash_search_index`. Cu ajutorul lui `perf script` se pot parsea evenimentele înregistrate în sample-uri în metoda `process_event`. Mai multe informații despre `perf script` se pot găsi la: `man perf-script-python` [<http://man7.org/linux/man-pages/man1/perf-script-python.1.html>] și exemplu de utilizare [<https://lwn.net/Articles/620900/>]

Verificați rezultatul utilizând comanda `perf report`.

## Exercițiul 2 - Row/Column major order

Folosind utilitarul `perf_3.2` dorim să determinăm dacă limbajul C este column-major sau row-major (row-major-order [[http://en.wikipedia.org/wiki/Row-major\\_order](http://en.wikipedia.org/wiki/Row-major_order)]).

Intrați în directorul `2-major` și completați programul `row.c` astfel încât să incrementeze elementele unei matrice pe linii, după care completați programul `columns.c` astfel încât să incrementeze elementele unei matrice pe coloane.

Determinați numărul de cache-miss-uri comparativ cu numărul de accese la cache folosind `perf stat` pentru a urmări evenimentul `L1-dcache-load-misses`. Pentru a vedea evenimentele disponibile folosiți comanda `perf list`. Folosiți opțiunea `-e` a utilitarului `perf` pentru a specifica un anumit eveniment de urmărit (revedeți secțiunea [perfcounters](#)).

## Exercițiul 3 - Busy

Intrați în directorul `3-busy` și inspectați fișierul `busy.c`. Rulați programul `busy` și analizați încărcarea sistemului folosind comanda `sudo perf top`. Ce funcție pare să încarce sistemul?



## Exercițiul 4 - Căutare într-un șir de caractere

Intrați în directorul 4-find-char/ și analizați conținutul fișierului find-char.c. Compilați fișierul find-char.c și rulați executabilul obținut.

Identificați, folosind perf record și perf report, care este funcția care ocupă cel mai mult timp de procesor și încercați să îmbunătățiți performanțele programului.

## Exercițiul 5 - Printing order

Intrați în directorul 5-print/ și analizați conținutul fișierului print.c. Folosiți comanda make print pentru a compila programul print. Există fișierul Makefile?

Care este ordinea în care se fac scrierile la consolă? Explicați output-ul.

Puneți o instrucțiune sleep(5) înainte de return 0; în funcția main și folosiți comanda strace -e write ./print pentru a găsi explicația.

## Exercițiul 6 - Flowers reloaded

Intrați în directorul 6-flowers/ și analizați conținutul fișierului flowers.c. Compilați fișierul flowers.c și rulați executabilul flowers. Ce se întâmplă? Folosiți valgrind cu opțiunea --tool=memcheck. Afișați valoarea celui de-al treilea element al array-ului flowers, adică flowers[2].

## Exercițiul 7 - Buffer overflow exploit

Rezolvați acest exercițiu pe mașina virtuală.

Intrați în directorul 7-exploit/ și analizați conținutul fișierului exploit.c. Folosiți comanda make pentru a compila executabilul exploit. Identificați o problemă în funcția read\_name.

Folosiți gdb pentru a investiga stiva înaintea efectuării apelului read.

```
student@spook:~$ gdb ./exploit
(gdb) break read_name
(gdb) run
```

Afișați adresele variabilelor name și access.

```
(gdb) print/x &access
(gdb) print/x &name
```

Observați că diferența între adresa variabilei access și adresa bufferului name este de 0x10 (16) octeți, ceea ce înseamnă că variabila access se află imediat la sfârșitul datelor din bufferul name.

Folosindu-vă de informațiile obținute, construiți un input convenabil pe care să îl oferiți executabilului exploit, astfel încât acesta să vă afișeze stringul "Good job, you hacked me!".

Pentru a genera caractere neprintabile, puteți folosi interpretorul Python: python -c.

```
student@spook:~$ python -c 'print "A"*8 + "\x01\x00\x00\x00" | ./exploit
```

Comanda de mai sus va genera 8 octeți cu valoarea 'A' (codul ASCII 0x41), un octet cu valoarea 0x01 și încă 3 octeți cu valoarea 0x00 și îi va oferi la stdin executabilului exploit. Rețineți că datele sunt

structurate în memorie în format little endian, prin urmare, dacă ultimii 4 octeți vor ajunge să suprascrie o adresă, aceasta va fi interpretată ca 0x00000001, NU 0x01000000.

## Exercițiul 8 - Trace the mystery

Intrați în directorul 8-mystery/ unde găsiți executabilul mystery. Investigați și explicați ce face acesta. Revedeți secțiunea strace.

## Soluții

---

- Soluții laborator 7 [<http://elf.cs.pub.ro/so/res/laboratoare/lab07-sol.zip>]

## Resurse utile

---

- GNU gprof manual [<http://sourceware.org/binutils/docs/gprof/>]
- linux/tools/perf [<http://lxr.linux.no/linux+v2.6.38/tools/perf/>]
- [Announce] Performance Counters for Linux, v8 [<http://lkml.org/lkml/2009/6/6/149>]
- Profiling tools and techniques [<http://www.pixelbeat.org/programming/profiling/>]
- Is Parallel Programming Hard, And, If So, What Can You Do About It? [<http://kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html>]
- gprof, Valgrind and gperftools – an evaluation of some tools for application level CPU profiling on Linux [<http://gernotklingler.com/blog/gprof-valgrind-gperftools-evaluation-tools-application-level-cpu-profiling-linux/>]
- Linux Profiling Tools and Techniques [<http://www.pixelbeat.org/programming/profiling/>]

so/laboratoare/laborator-07.txt · Last modified: 2019/04/10 09:47 by razvan.nitu1305