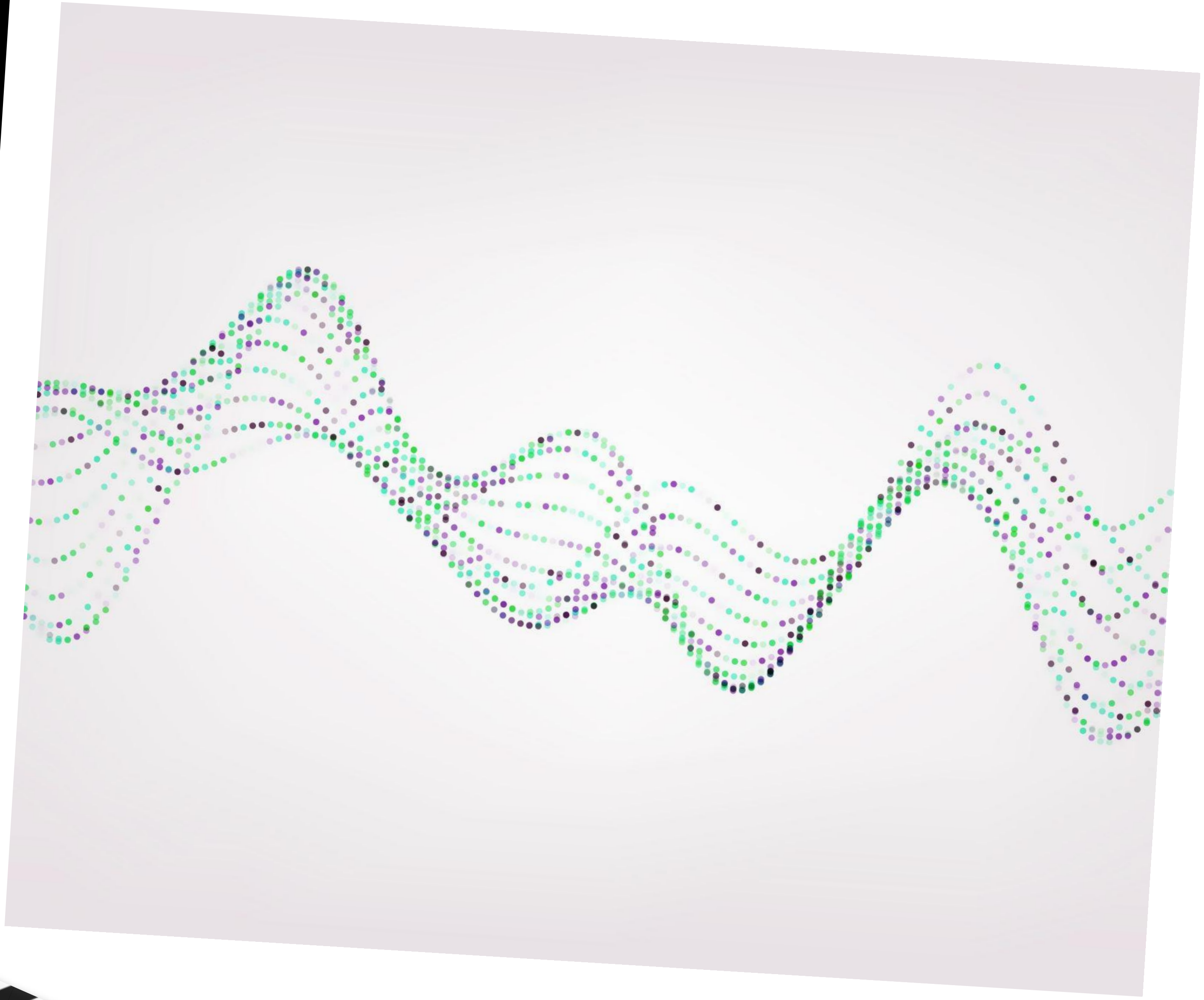
Code Profiling & Optimization

Day 4

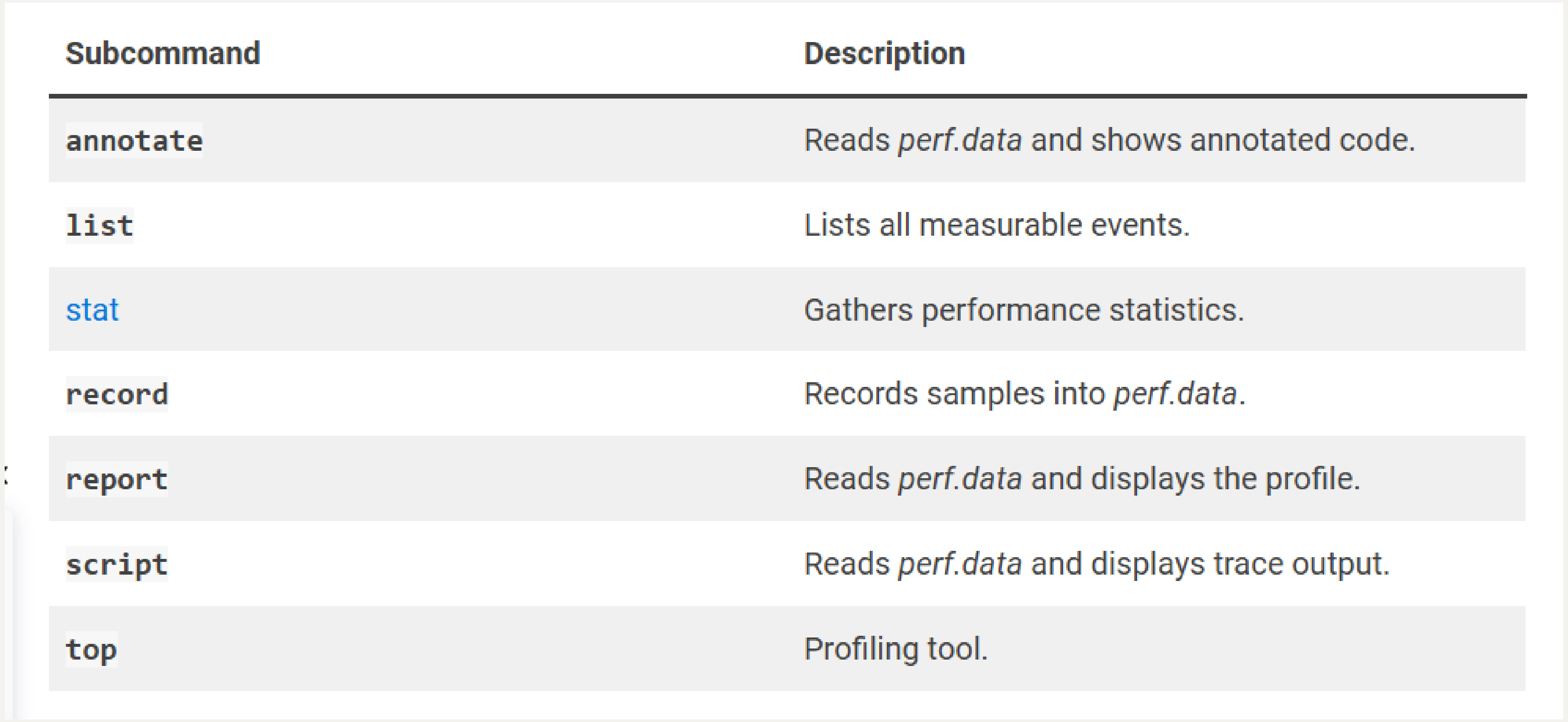
Measuring Performance Stats with Linux Perf

Subhrajit & Pratyush



Getting started with Linux Perf

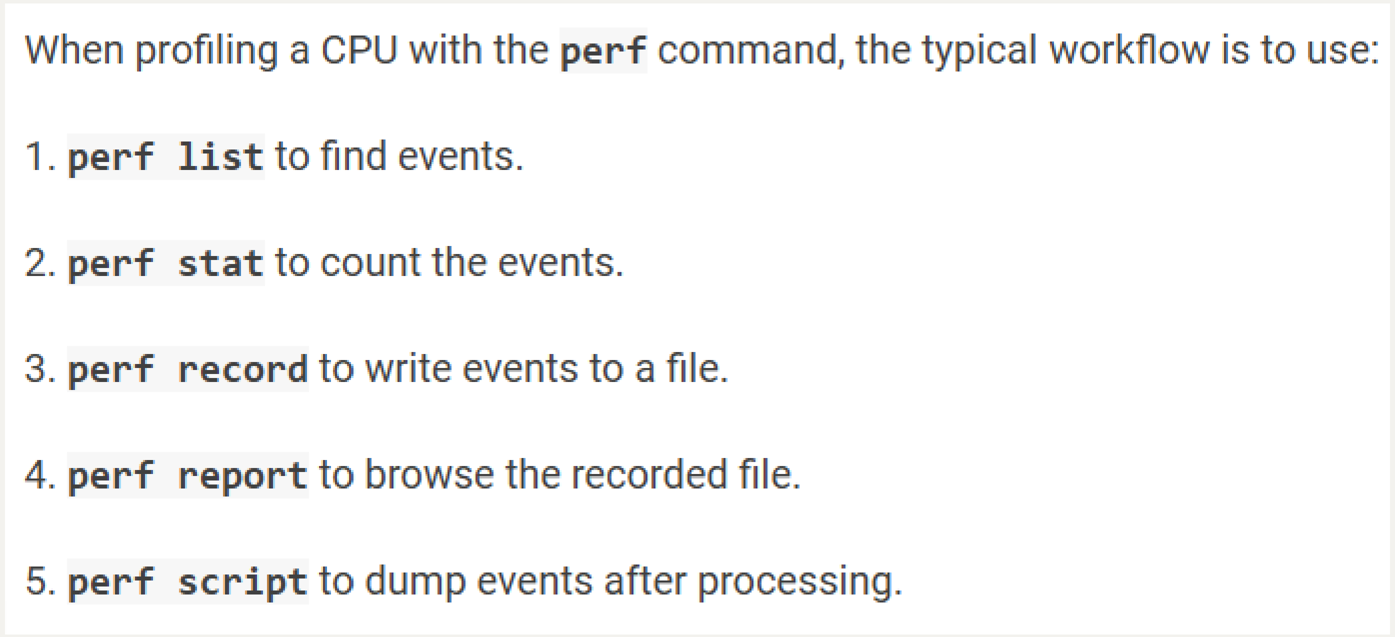
- a lightweight command-line utility for profiling and monitoring CPU performance on Linux systems.
- The perf command, by default, requires sudo privileges.
- `perf <options> subcommand <options/arguments>`



Subcommand	Description
`annotate`	Reads *perf.data* and shows annotated code.
`list`	Lists all measurable events.
stat	Gathers performance statistics.
`record`	Records samples into *perf.data*.
`report`	Reads *perf.data* and displays the profile.
`script`	Reads *perf.data* and displays trace output.
`top`	Profiling tool.

Getting started with Linux Perf

```
perf <options> subcommand <options/arguments>
```



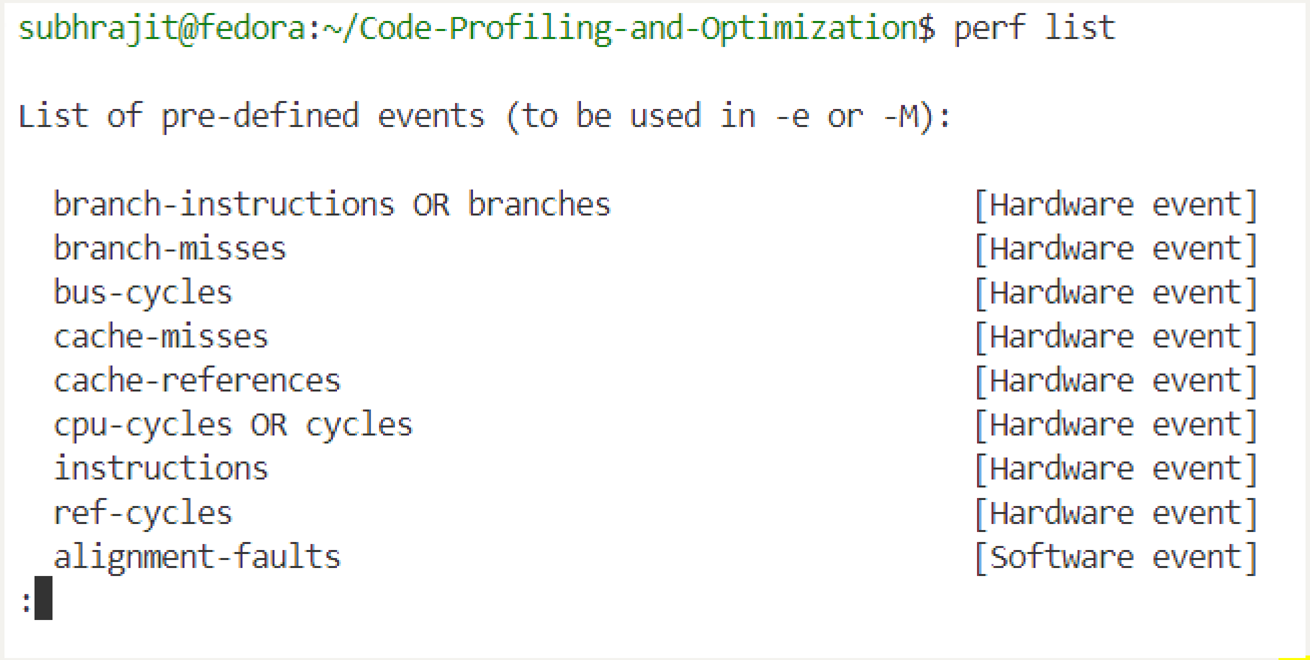
When profiling a CPU with the `perf` command, the typical workflow is to use:

1. `perf list` to find events.
2. `perf stat` to count the events.
3. `perf record` to write events to a file.
4. `perf report` to browse the recorded file.
5. `perf script` to dump events after processing.

Getting started with Linux Perf

```
perf <options> subcommand <options/arguments>
```

1. `perf list` to find events.



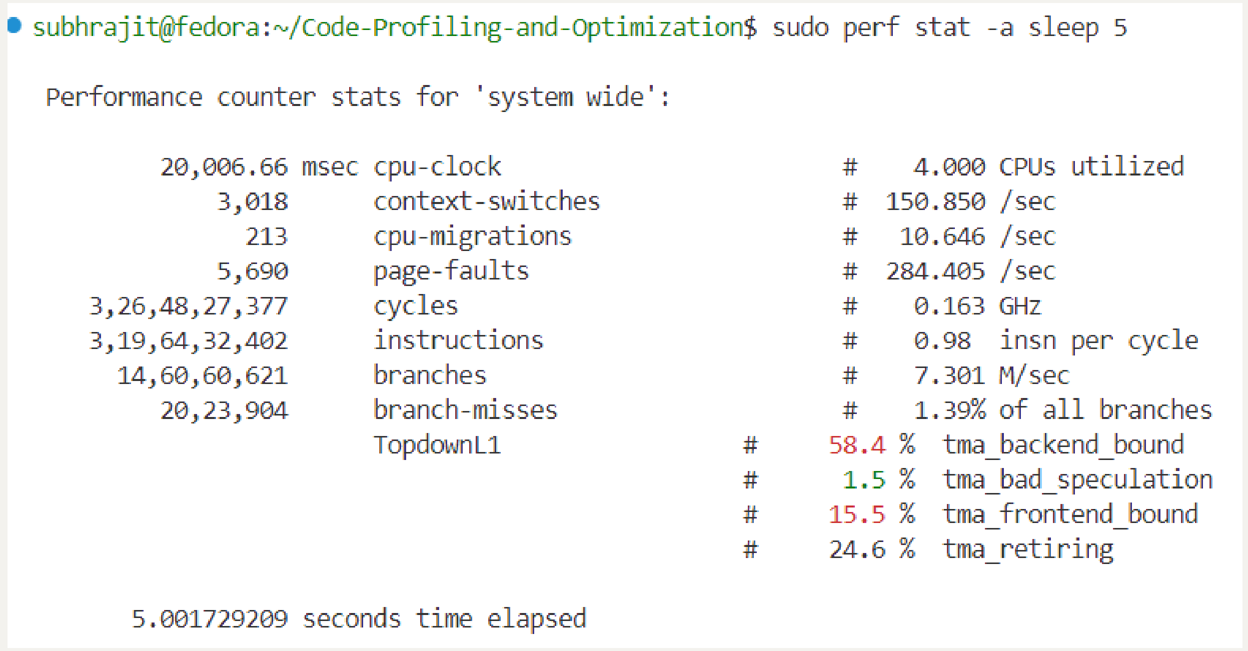
```
subhrajit@fedora:~/Code-Profiling-and-Optimization$ perf list

List of pre-defined events (to be used in -e or -M):

  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  cache-misses                                       [Hardware event]
  cache-references                                   [Hardware event]
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  ref-cycles                                         [Hardware event]
  alignment-faults                                   [Software event]
:
```

Perf list

```
perf <options> subcommand <options/arguments>
```

2. **`perf stat`** to count the events.

To display CPU performance statistics for all standard CPU-wide hardware and software events, run:

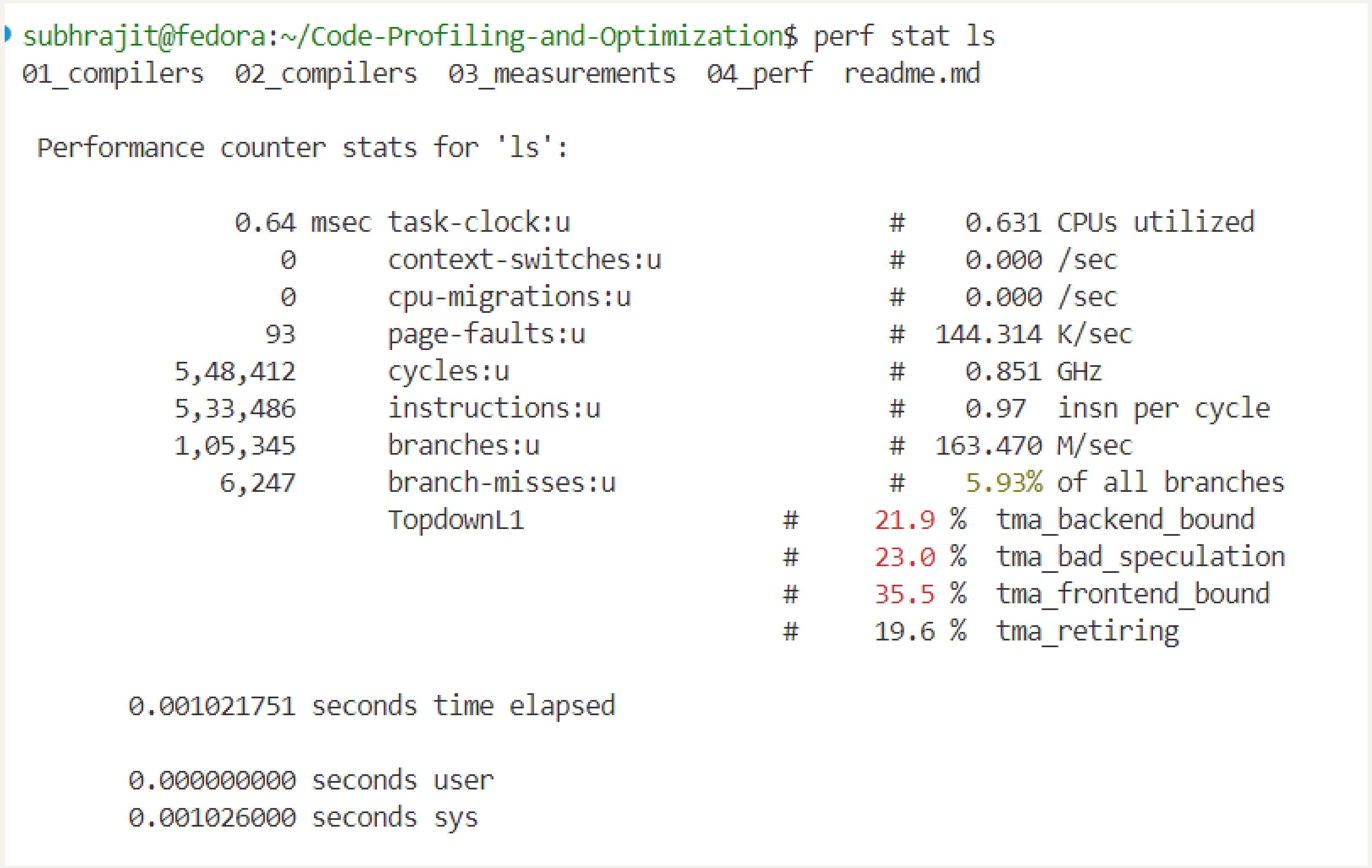


```
● subhrajit@fedora:~/Code-Profiling-and-Optimization$ sudo perf stat -a sleep 5

 Performance counter stats for 'system wide':

         20,006.66 msec cpu-clock                        #    4.000 CPUs utilized
             3,018      context-switches                 #  150.850 /sec
               213      cpu-migrations                   #   10.646 /sec
             5,690      page-faults                      #  284.405 /sec
    3,26,48,27,377      cycles                           #    0.163 GHz
    3,19,64,32,402      instructions                     #    0.98  insn per cycle
      14,60,60,621      branches                         #    7.301 M/sec
         20,23,904      branch-misses                    #    1.39% of all branches
                        TopdownL1                 #     58.4 %  tma_backend_bound
                                                  #      1.5 %  tma_bad_speculation
                                                  #     15.5 %  tma_frontend_bound
                                                  #     24.6 %  tma_retiring

       5.001729209 seconds time elapsed
```

Perf stat

`perf <options> subcommand <options/arguments>`

2. `perf stat` to count the events.

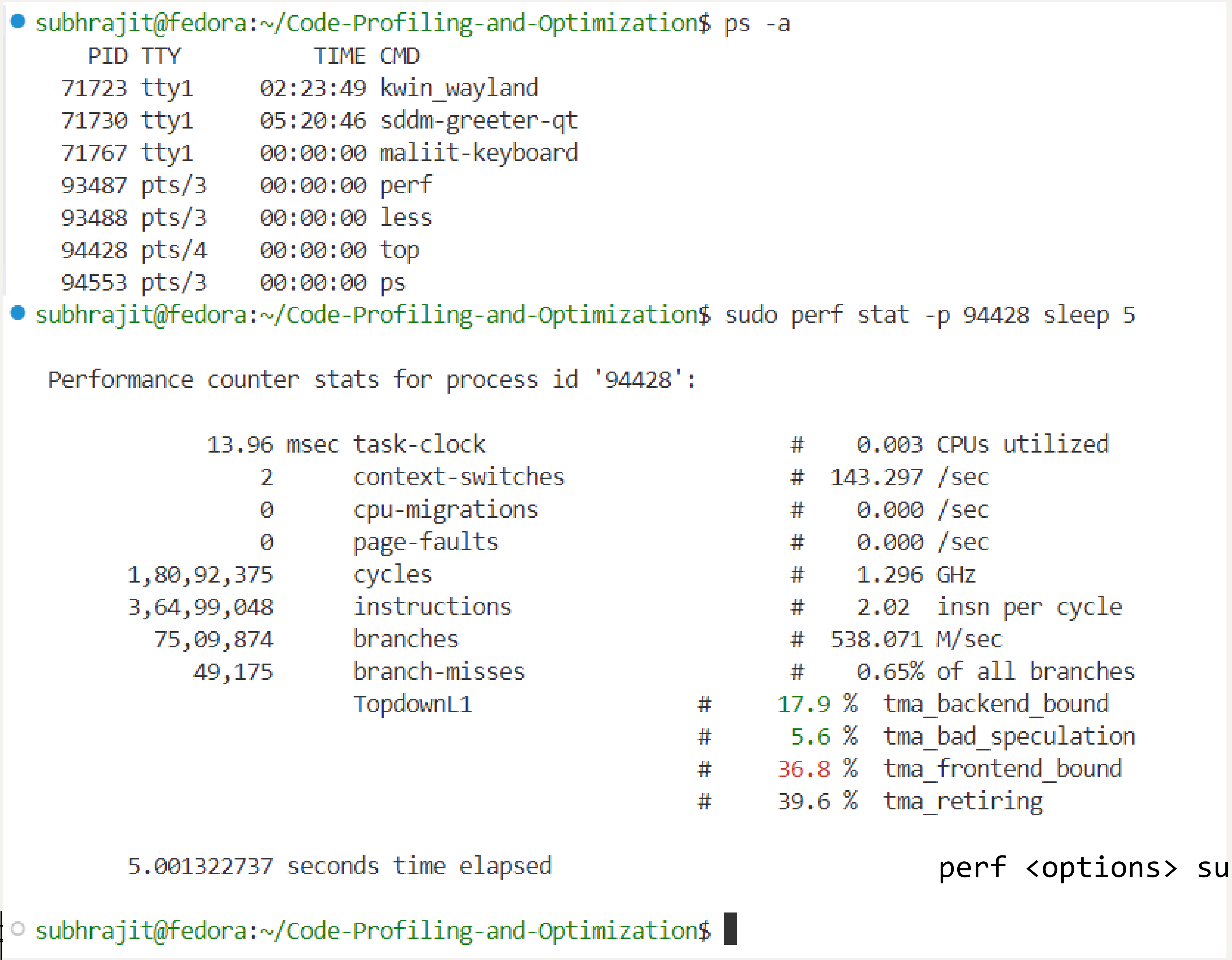


```
subhrajit@fedora:~/Code-Profiling-and-Optimization$ perf stat ls
01_compilers  02_compilers  03_measurements  04_perf  readme.md

 Performance counter stats for 'ls':

              0.64 msec task-clock:u                     #    0.631 CPUs utilized
                 0      context-switches:u               #    0.000 /sec
                 0      cpu-migrations:u                 #    0.000 /sec
                93      page-faults:u                    #  144.314 K/sec
          5,48,412      cycles:u                         #    0.851 GHz
          5,33,486      instructions:u                   #    0.97  insn per cycle
          1,05,345      branches:u                       #  163.470 M/sec
             6,247      branch-misses:u                  #    5.93% of all branches
                        TopdownL1                 #     21.9 %  tma_backend_bound
                                                  #     23.0 %  tma_bad_speculation
                                                  #     35.5 %  tma_frontend_bound
                                                  #     19.6 %  tma_retiring

       0.001021751 seconds time elapsed

       0.000000000 seconds user
       0.001026000 seconds sys
```

Perf stat

`perf <options> subcommand <options/arguments>`

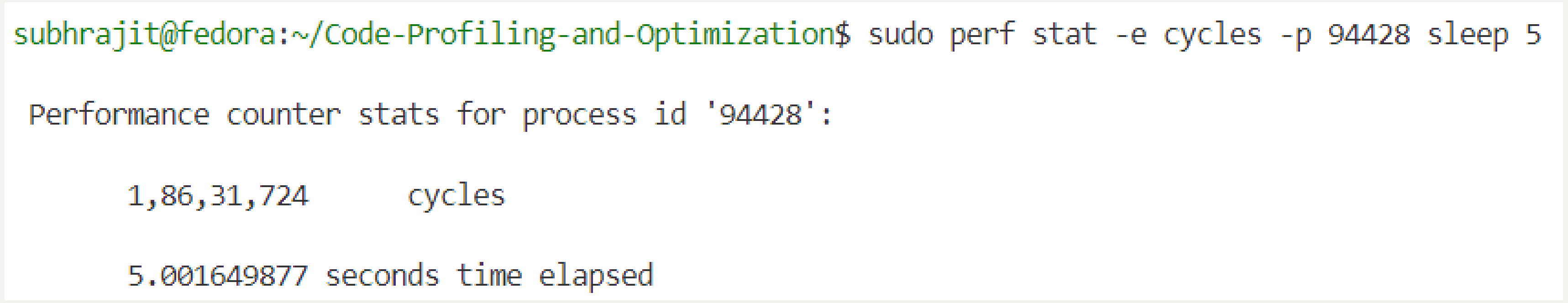
2. **`perf stat`** to count the events.



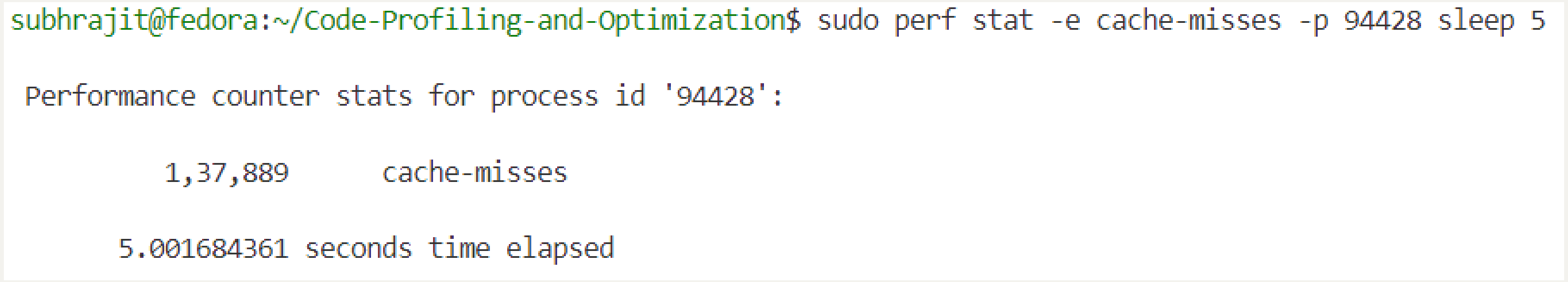
```
● subhrajit@fedora:~/Code-Profiling-and-Optimization$ ps -a
    PID TTY          TIME CMD
  71723 tty1     02:23:49 kwin_wayland
  71730 tty1     05:20:46 sddm-greeter-qt
  71767 tty1     00:00:00 maliit-keyboard
  93487 pts/3    00:00:00 perf
  93488 pts/3    00:00:00 less
  94428 pts/4    00:00:00 top
  94553 pts/3    00:00:00 ps
● subhrajit@fedora:~/Code-Profiling-and-Optimization$ sudo perf stat -p 94428 sleep 5

 Performance counter stats for process id '94428':

             13.96 msec task-clock                       #    0.003 CPUs utilized
                 2      context-switches                 #  143.297 /sec
                 0      cpu-migrations                   #    0.000 /sec
                 0      page-faults                      #    0.000 /sec
       1,80,92,375      cycles                           #    1.296 GHz
       3,64,99,048      instructions                     #    2.02  insn per cycle
         75,09,874      branches                         #  538.071 M/sec
            49,175      branch-misses                    #    0.65% of all branches
                        TopdownL1                 #     17.9 %  tma_backend_bound
                                                  #      5.6 %  tma_bad_speculation
                                                  #     36.8 %  tma_frontend_bound
                                                  #     39.6 %  tma_retiring

       5.001322737 seconds time elapsed

○ subhrajit@fedora:~/Code-Profiling-and-Optimization$ █
```

Perf stat

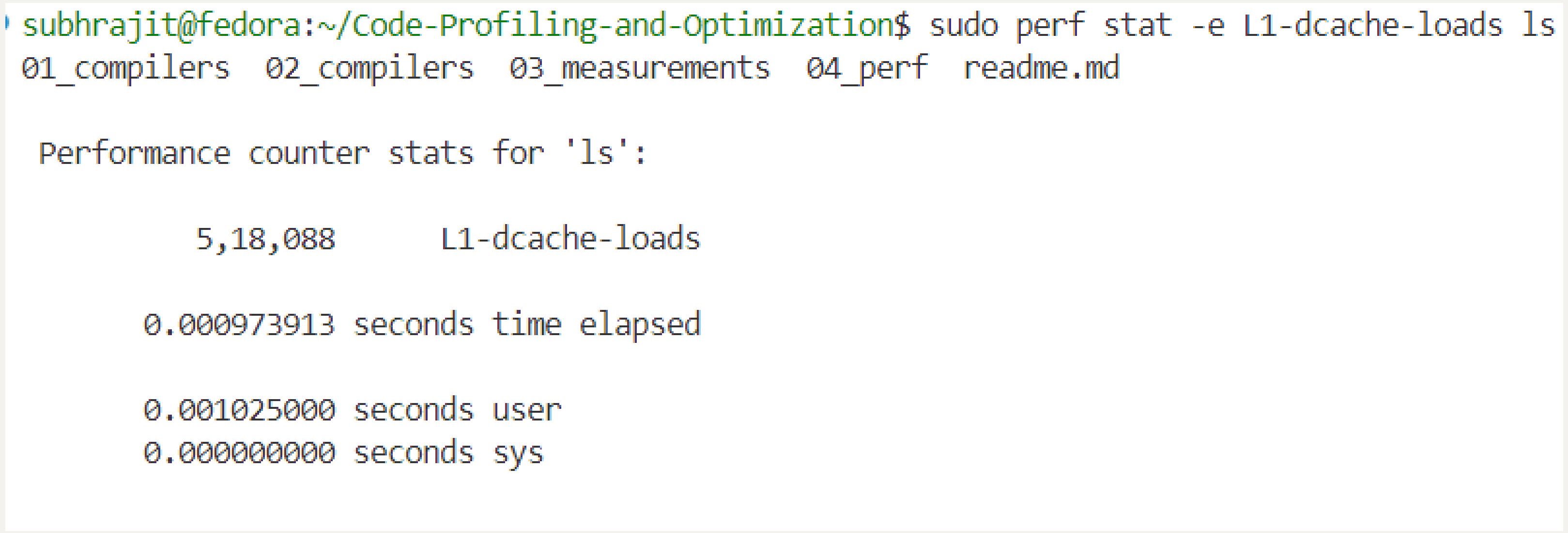
```
perf <options> subcommand <options/arguments>
```



```
subhrajit@fedora:~/Code-Profiling-and-Optimization$ sudo perf stat -e cycles -p 94428 sleep 5

 Performance counter stats for process id '94428':

       1,86,31,724      cycles

       5.001649877 seconds time elapsed
```



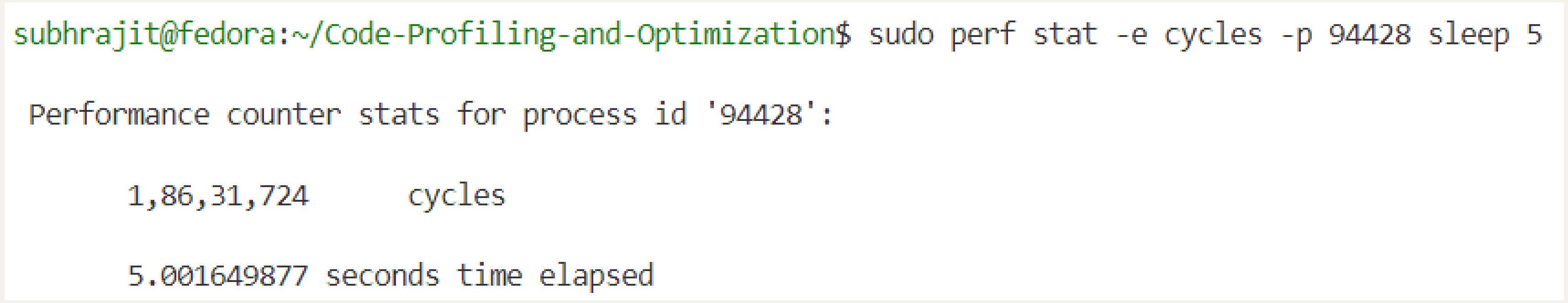
```
subhrajit@fedora:~/Code-Profiling-and-Optimization$ sudo perf stat -e cache-misses -p 94428 sleep 5

 Performance counter stats for process id '94428':

          1,37,889      cache-misses

       5.001684361 seconds time elapsed
```



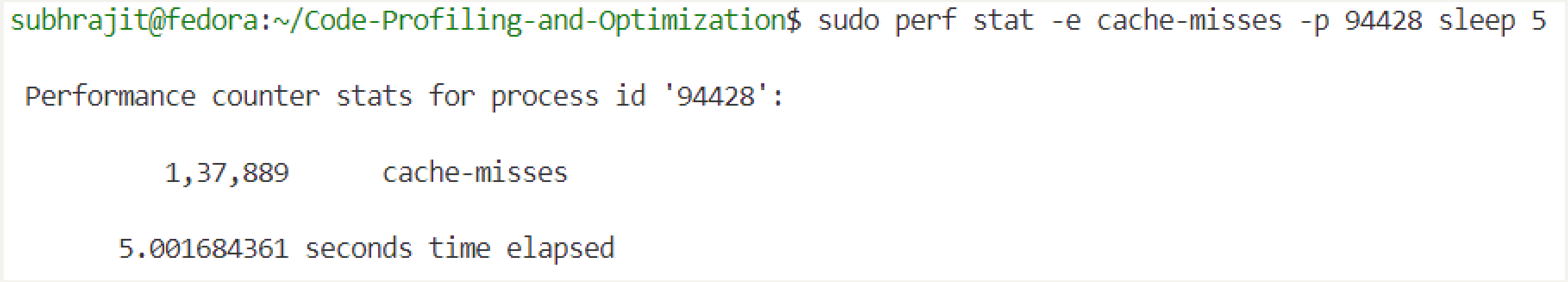
```
subhrajit@fedora:~/Code-Profiling-and-Optimization$ sudo perf stat -e L1-dcache-loads ls
01_compilers  02_compilers  03_measurements  04_perf  readme.md

 Performance counter stats for 'ls':

          5,18,088      L1-dcache-loads

       0.000973913 seconds time elapsed

       0.001025000 seconds user
       0.000000000 seconds sys
```

Perf stat

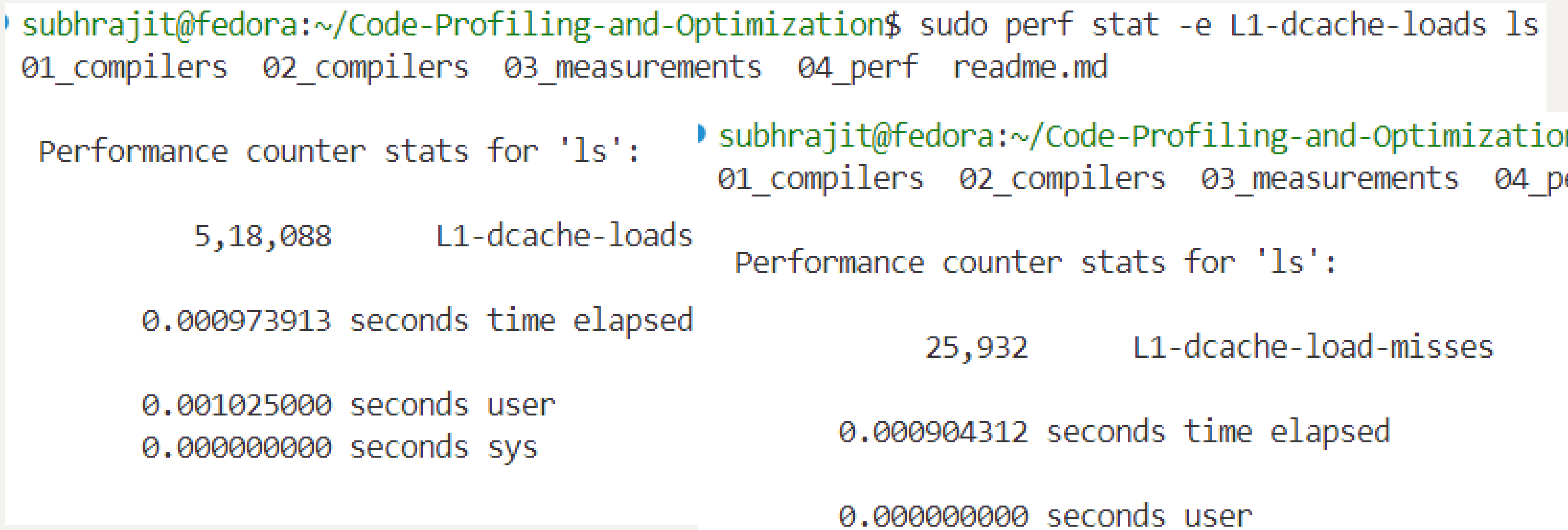
`perf <options> subcommand <options/arguments>`



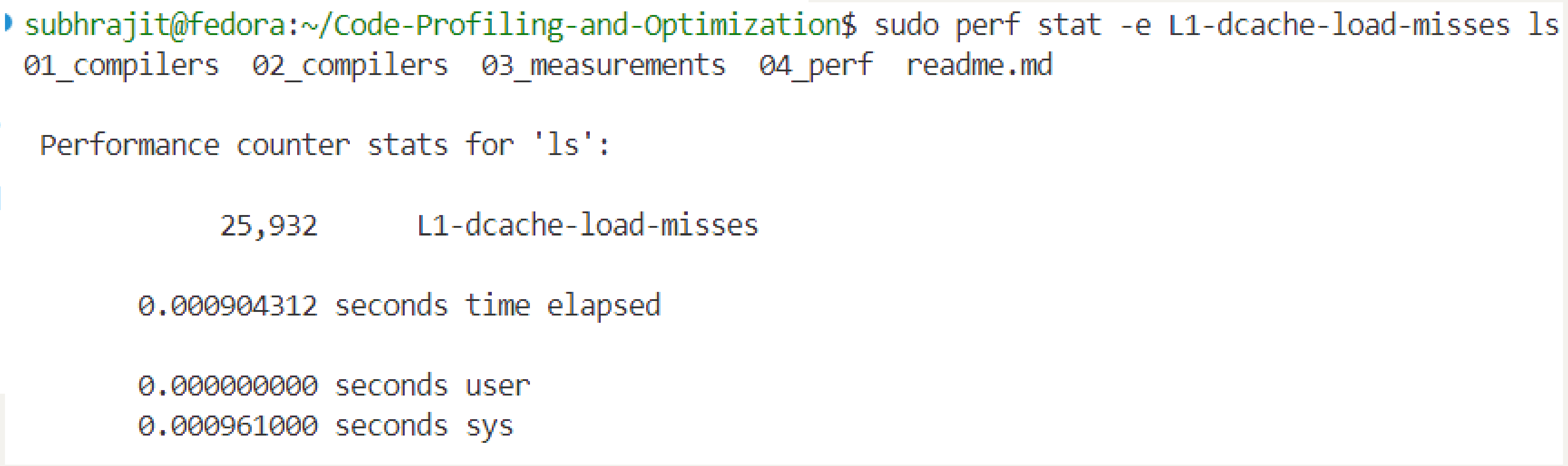
```
subhrajit@fedora:~/Code-Profiling-and-Optimization$ sudo perf stat -e cycles -p 94428 sleep 5

 Performance counter stats for process id '94428':

       1,86,31,724      cycles

       5.001649877 seconds time elapsed
```



```
subhrajit@fedora:~/Code-Profiling-and-Optimization$ sudo perf stat -e cache-misses -p 94428 sleep 5

 Performance counter stats for process id '94428':

          1,37,889      cache-misses

       5.001684361 seconds time elapsed
```



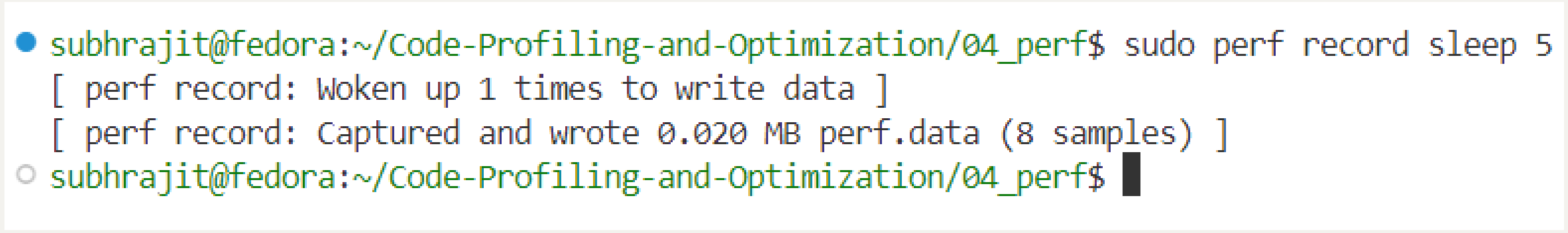
```
subhrajit@fedora:~/Code-Profiling-and-Optimization$ sudo perf stat -e L1-dcache-loads ls
01_compilers  02_compilers  03_measurements  04_perf  readme.md

 Performance counter stats for 'ls':

          5,18,088      L1-dcache-loads

       0.000973913 seconds time elapsed

       0.001025000 seconds user
       0.000000000 seconds sys
```



```
subhrajit@fedora:~/Code-Profiling-and-Optimization$ sudo perf stat -e L1-dcache-load-misses ls
01_compilers  02_compilers  03_measurements  04_perf  readme.md

 Performance counter stats for 'ls':

            25,932      L1-dcache-load-misses

       0.000904312 seconds time elapsed

       0.000000000 seconds user
       0.000961000 seconds sys
```

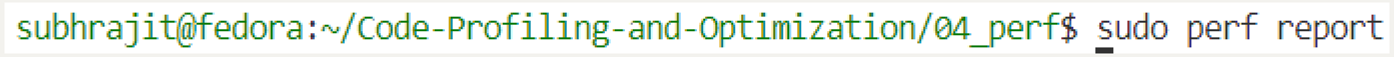
Perf record

3. **perf record** to write events to a file.

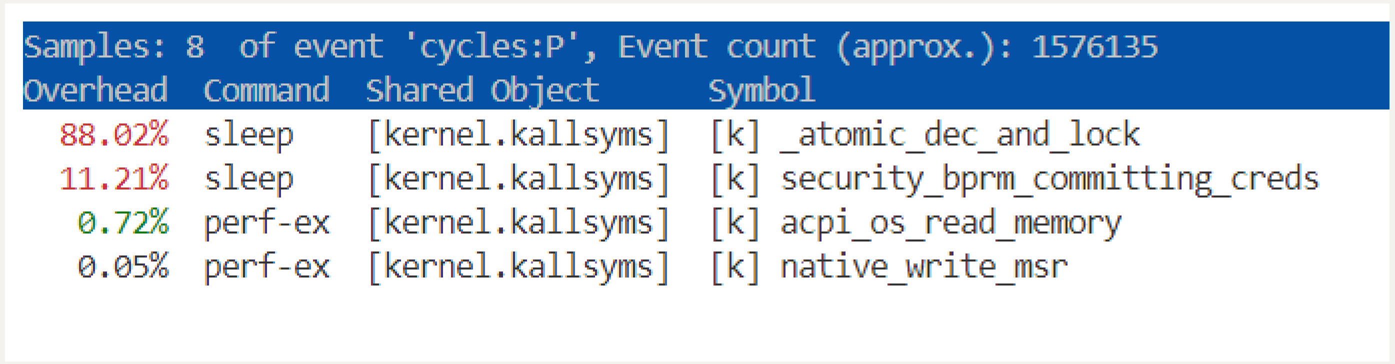


```
● subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf$ sudo perf record sleep 5
  [ perf record: Woken up 1 times to write data ]
  [ perf record: Captured and wrote 0.020 MB perf.data (8 samples) ]
○ subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf$ 
```

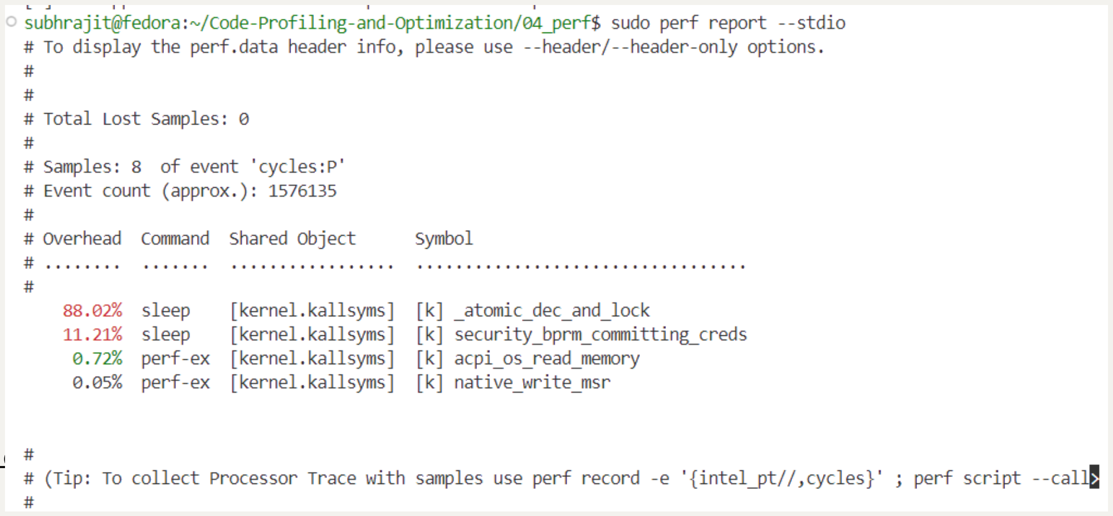
4. **perf report** to browse the recorded file.



```
subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf$ sudo perf report
```

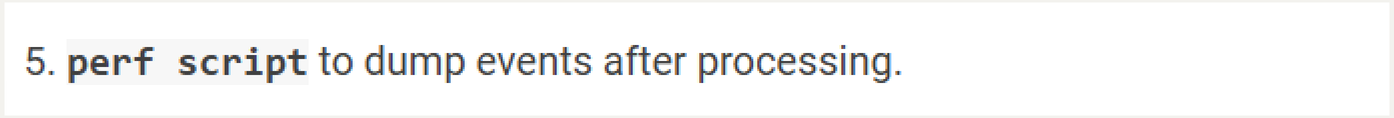


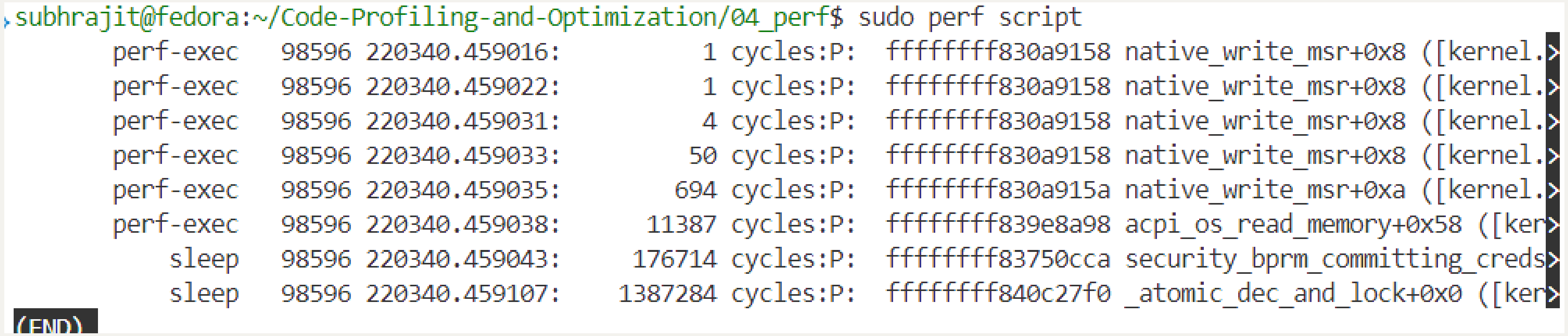
```
Samples: 8  of event 'cycles:P', Event count (approx.): 1576135
Overhead  Command  Shared Object       Symbol
  88.02%  sleep    [kernel.kallsyms]  [k] _atomic_dec_and_lock
  11.21%  sleep    [kernel.kallsyms]  [k] security_bprm_committing_creds
   0.72%  perf-ex  [kernel.kallsyms]  [k] acpi_os_read_memory
   0.05%  perf-ex  [kernel.kallsyms]  [k] native_write_msr
```



```
○ subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf$ sudo perf report --stdio
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 8  of event 'cycles:P'
# Event count (approx.): 1576135
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................  ...................................
#
    88.02%  sleep    [kernel.kallsyms]  [k] _atomic_dec_and_lock
    11.21%  sleep    [kernel.kallsyms]  [k] security_bprm_committing_creds
     0.72%  perf-ex  [kernel.kallsyms]  [k] acpi_os_read_memory
     0.05%  perf-ex  [kernel.kallsyms]  [k] native_write_msr


#
# (Tip: To collect Processor Trace with samples use perf record -e '{intel_pt//,cycles}' ; perf script --call
#
```

`perf <options> subcommand <options/arguments>`


5. **perf script** to dump events after processing.

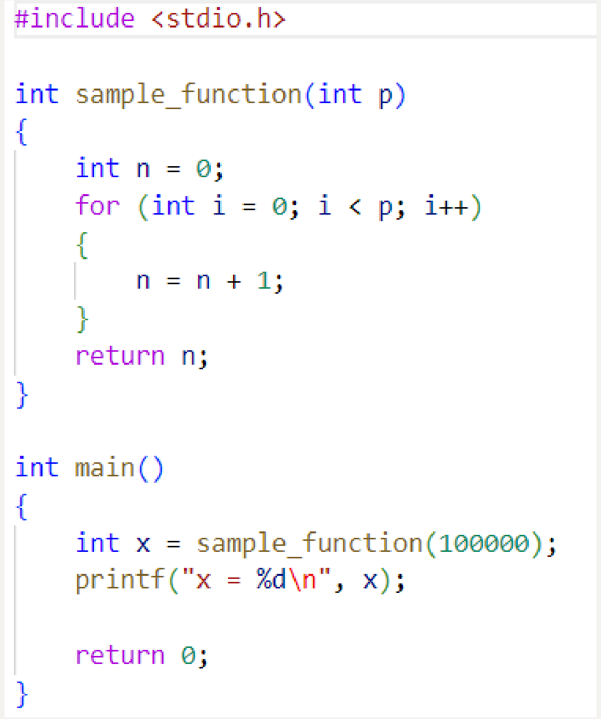

```
subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf$ sudo perf script
       perf-exec   98596 220340.459016:          1 cycles:P:  ffffffff830a9158 native_write_msr+0x8 ([kernel.>
       perf-exec   98596 220340.459022:          1 cycles:P:  ffffffff830a9158 native_write_msr+0x8 ([kernel.>
       perf-exec   98596 220340.459031:          4 cycles:P:  ffffffff830a9158 native_write_msr+0x8 ([kernel.>
       perf-exec   98596 220340.459033:         50 cycles:P:  ffffffff830a9158 native_write_msr+0x8 ([kernel.>
       perf-exec   98596 220340.459035:        694 cycles:P:  ffffffff830a915a native_write_msr+0xa ([kernel.>
       perf-exec   98596 220340.459038:      11387 cycles:P:  ffffffff839e8a98 acpi_os_read_memory+0x58 ([ker>
           sleep   98596 220340.459043:     176714 cycles:P:  ffffffff83750cca security_bprm_committing_creds>
           sleep   98596 220340.459107:    1387284 cycles:P:  ffffffff840c27f0 _atomic_dec_and_lock+0x0 ([ker>
(END)
```

The output prints the perf.data details in time order. Use the script subcommand as post-processing data.

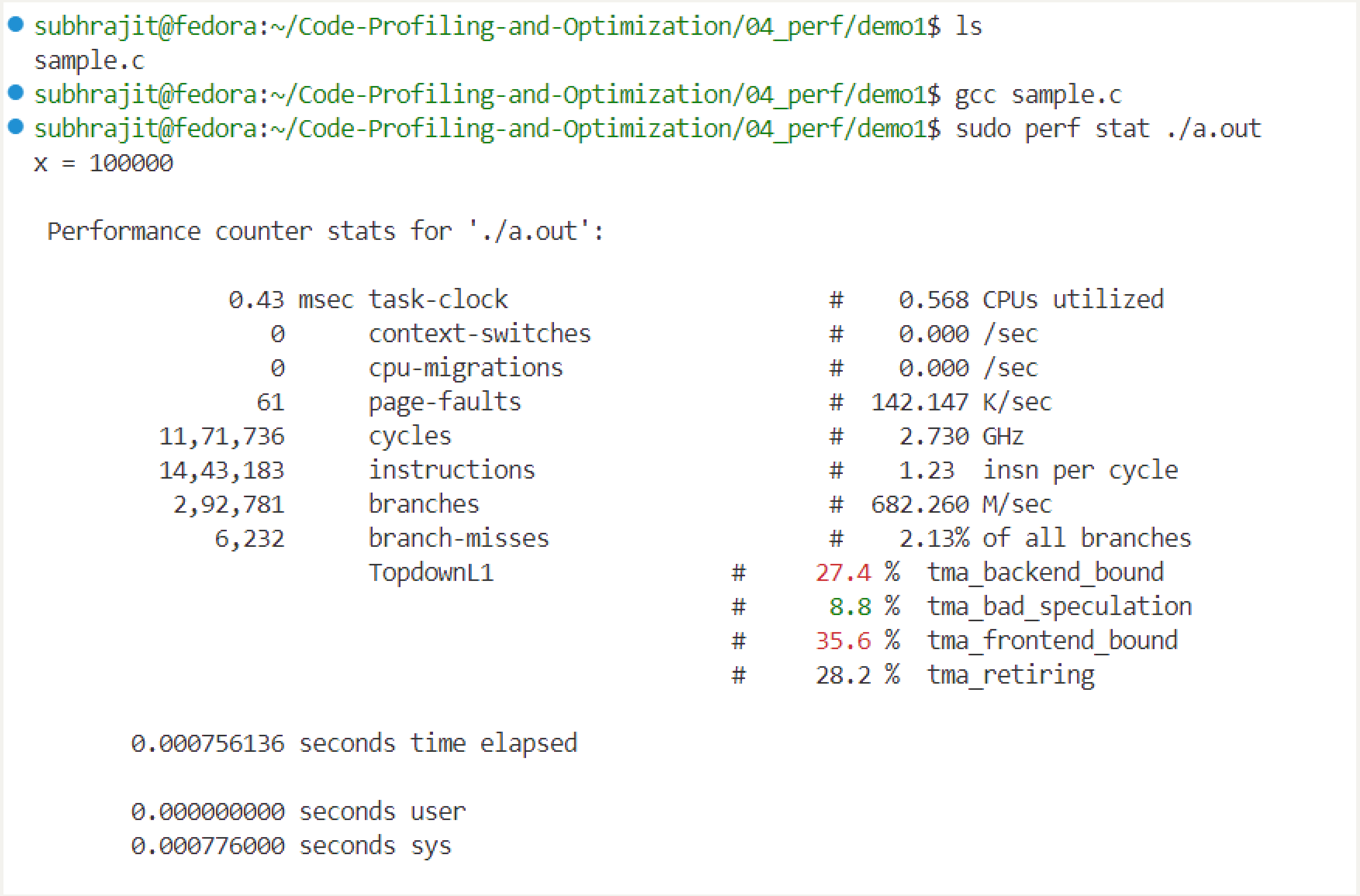
Perf script

perf <options> subcommand <options/arguments>

Sample Code for Perf

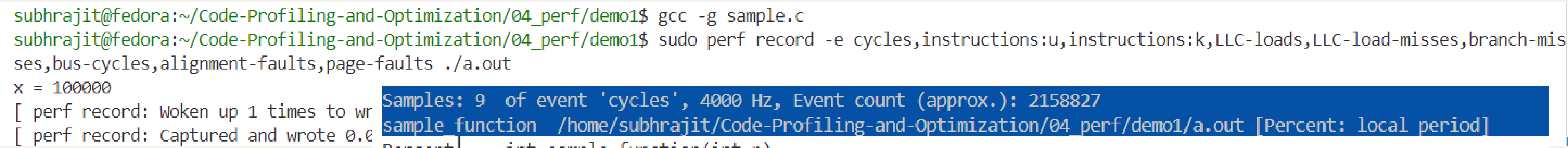


```c
#include <stdio.h>

int sample_function(int p)
{
    int n = 0;
    for (int i = 0; i < p; i++)
    {
        n = n + 1;
    }
    return n;
}

int main()
{
    int x = sample_function(100000);
    printf("x = %d\n", x);

    return 0;
}
```

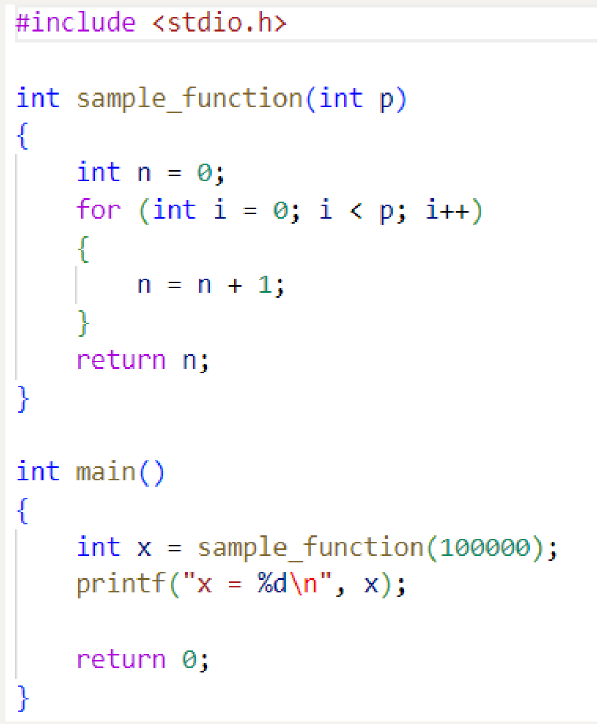


```
● subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ ls
  sample.c
● subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ gcc sample.c
● subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ sudo perf stat ./a.out
  x = 100000

   Performance counter stats for './a.out':

                0.43 msec task-clock                       #    0.568 CPUs utilized
                   0      context-switches                 #    0.000 /sec
                   0      cpu-migrations                   #    0.000 /sec
                  61      page-faults                      #  142.147 K/sec
           11,71,736      cycles                           #    2.730 GHz
           14,43,183      instructions                     #    1.23  insn per cycle
            2,92,781      branches                         #  682.260 M/sec
               6,232      branch-misses                    #    2.13% of all branches
                          TopdownL1                 #     27.4 %  tma_backend_bound
                                                    #      8.8 %  tma_bad_speculation
                                                    #     35.6 %  tma_frontend_bound
                                                    #     28.2 %  tma_retiring

         0.000756136 seconds time elapsed

         0.000000000 seconds user
         0.000776000 seconds sys
```

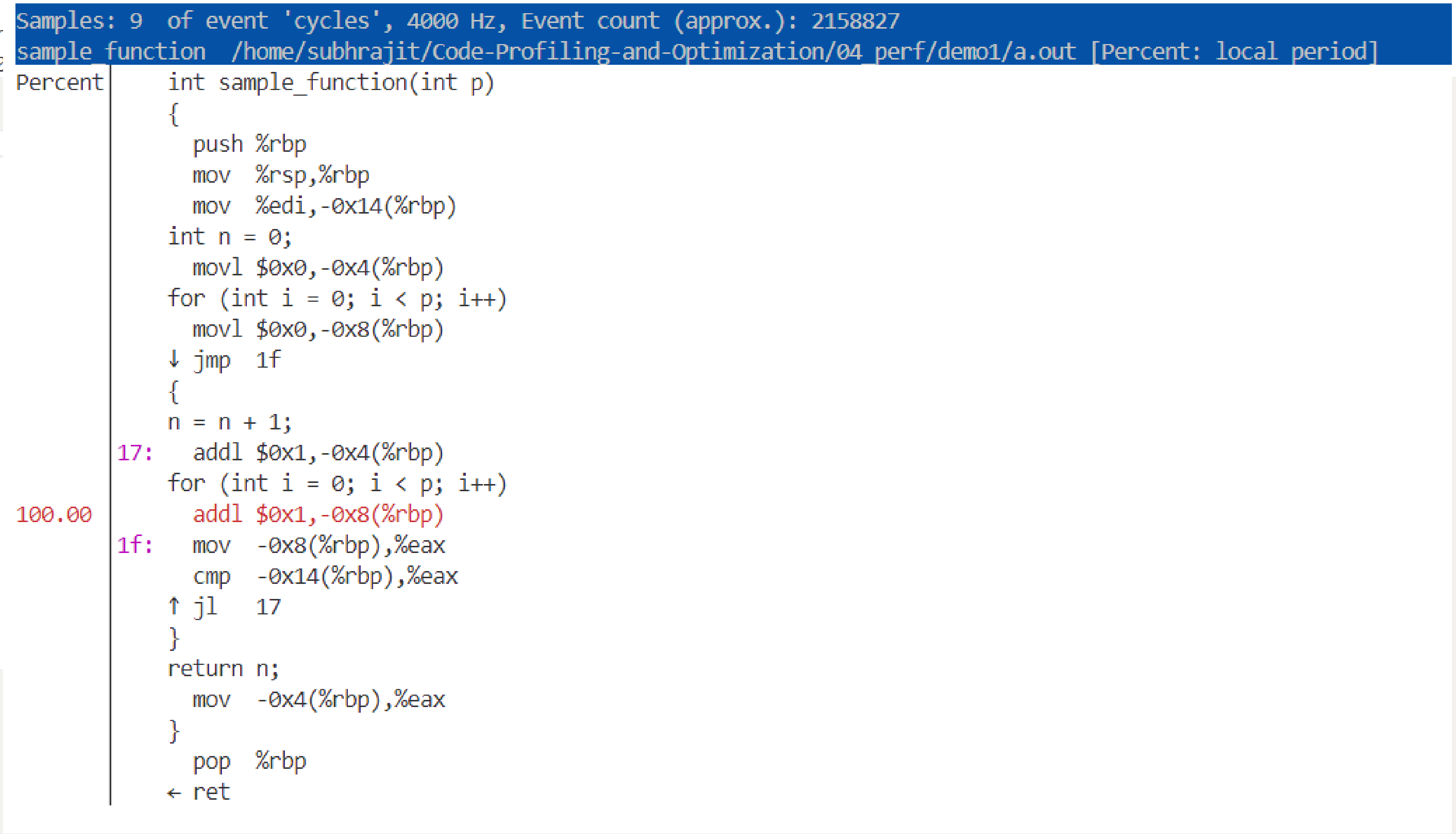
Sample Code for Perf



```
subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ gcc -g sample.c
subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ sudo perf record -e cycles,instructions:u,instructions:k,LLC-loads,LLC-load-misses,branch-mis
ses,bus-cycles,alignment-faults,page-faults ./a.out
x = 100000
[ perf record: Woken up 1 times to wr
[ perf record: Captured and wrote 0.0
```

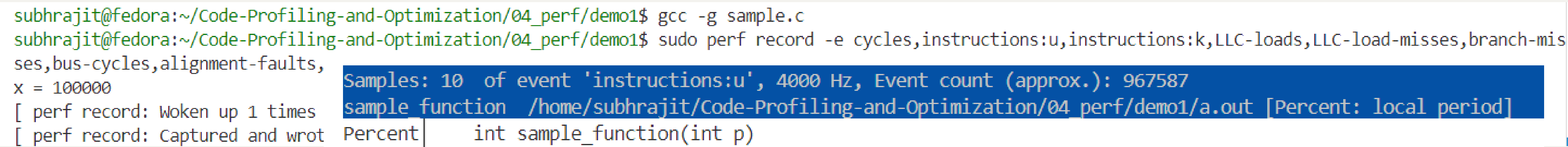


```c
#include <stdio.h>

int sample_function(int p)
{
    int n = 0;
    for (int i = 0; i < p; i++)
    {
        n = n + 1;
    }
    return n;
}

int main()
{
    int x = sample_function(100000);
    printf("x = %d\n", x);

    return 0;
}
```

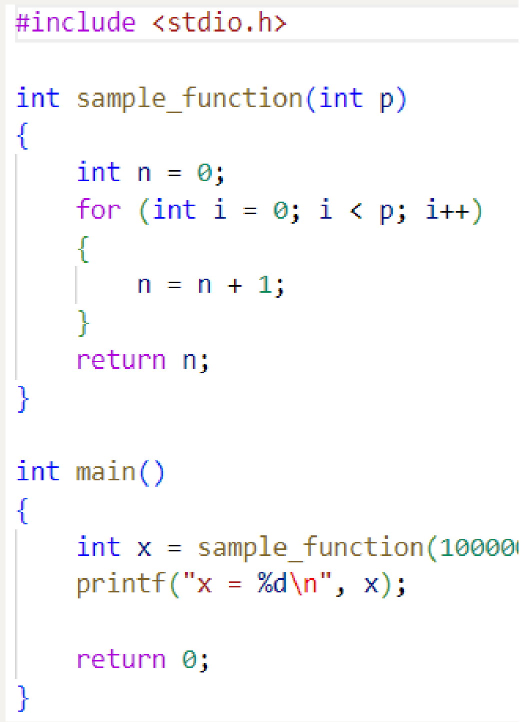


```
Samples: 9  of event 'cycles', 4000 Hz, Event count (approx.): 2158827
sample_function  /home/subhrajit/Code-Profiling-and-Optimization/04_perf/demo1/a.out [Percent: local period]
Percent │     int sample_function(int p)
        │     {
        │       push %rbp
        │       mov  %rsp,%rbp
        │       mov  %edi,-0x14(%rbp)
        │     int n = 0;
        │       movl $0x0,-0x4(%rbp)
        │     for (int i = 0; i < p; i++)
        │       movl $0x0,-0x8(%rbp)
        │     ↓ jmp  1f
        │     {
        │     n = n + 1;
        │17:    addl $0x1,-0x4(%rbp)
        │     for (int i = 0; i < p; i++)
100.00  │       addl $0x1,-0x8(%rbp)
        │1f:    mov  -0x8(%rbp),%eax
        │       cmp  -0x14(%rbp),%eax
        │     ↑ jl   17
        │     }
        │     return n;
        │       mov  -0x4(%rbp),%eax
        │     }
        │       pop  %rbp
        │     ← ret
```

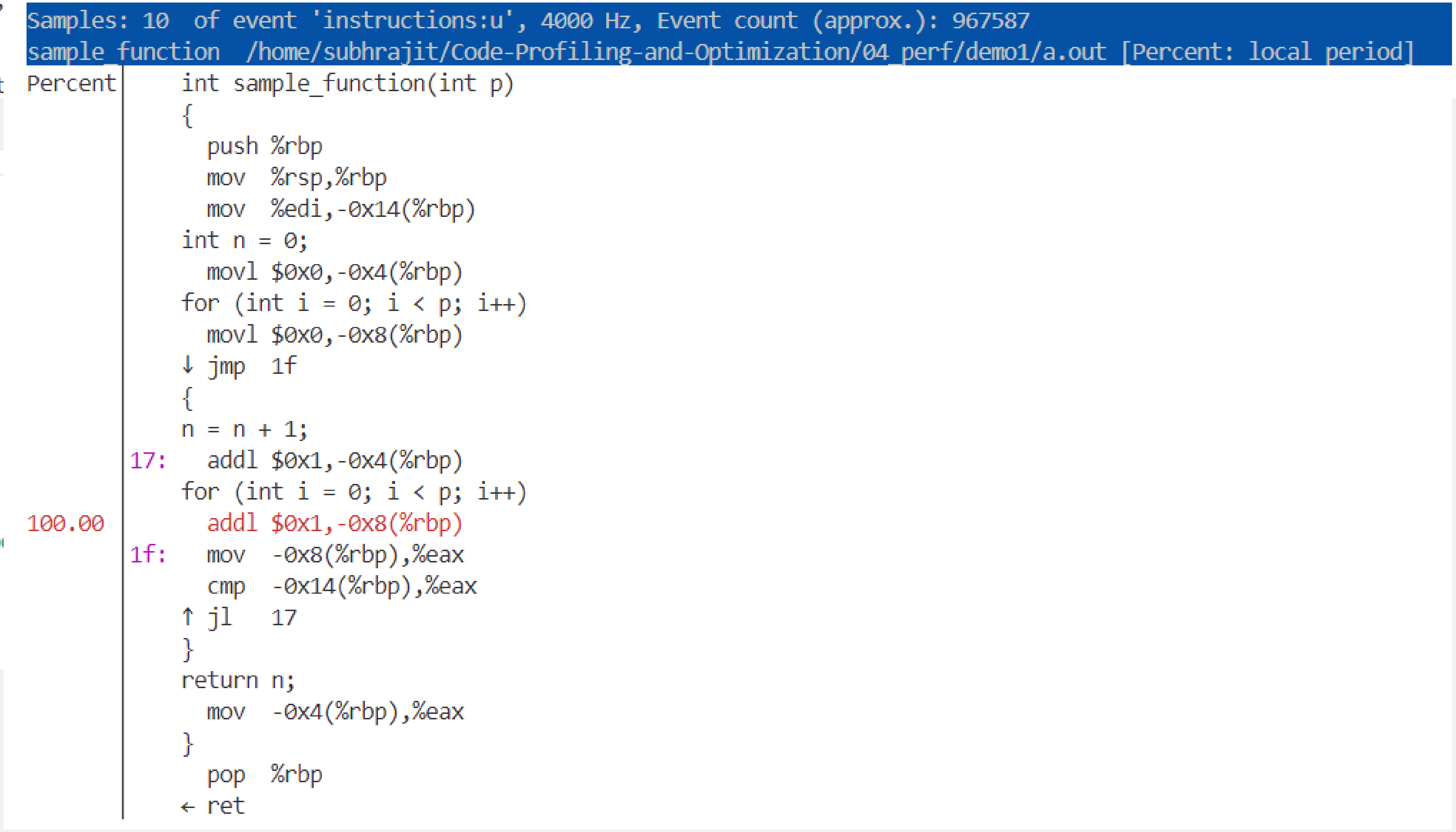
Sample Code for Perf



```
subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ gcc -g sample.c
subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ sudo perf record -e cycles,instructions:u,instructions:k,LLC-loads,LLC-load-misses,branch-mis
ses,bus-cycles,alignment-faults,
x = 100000
[ perf record: Woken up 1 times
[ perf record: Captured and wrot
```

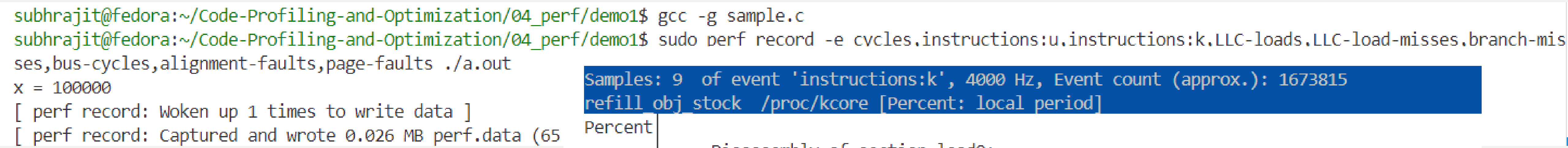


```c
#include <stdio.h>

int sample_function(int p)
{
    int n = 0;
    for (int i = 0; i < p; i++)
    {
        n = n + 1;
    }
    return n;
}

int main()
{
    int x = sample_function(10000
    printf("x = %d\n", x);

    return 0;
}
```

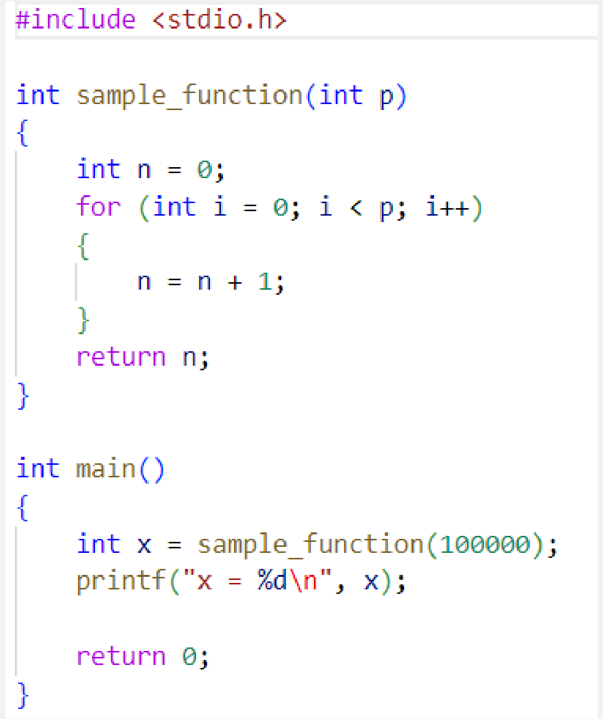


```
Samples: 10  of event 'instructions:u', 4000 Hz, Event count (approx.): 967587
sample_function  /home/subhrajit/Code-Profiling-and-Optimization/04_perf/demo1/a.out [Percent: local period]
Percent │      int sample_function(int p)
        │      {
        │        push %rbp
        │        mov  %rsp,%rbp
        │        mov  %edi,-0x14(%rbp)
        │      int n = 0;
        │        movl $0x0,-0x4(%rbp)
        │      for (int i = 0; i < p; i++)
        │        movl $0x0,-0x8(%rbp)
        │      ↓ jmp  1f
        │      {
        │      n = n + 1;
        │17:     addl $0x1,-0x4(%rbp)
        │      for (int i = 0; i < p; i++)
 100.00 │        addl $0x1,-0x8(%rbp)
        │1f:     mov  -0x8(%rbp),%eax
        │        cmp  -0x14(%rbp),%eax
        │      ↑ jl   17
        │      }
        │      return n;
        │        mov  -0x4(%rbp),%eax
        │      }
        │        pop  %rbp
        │      ← ret
```

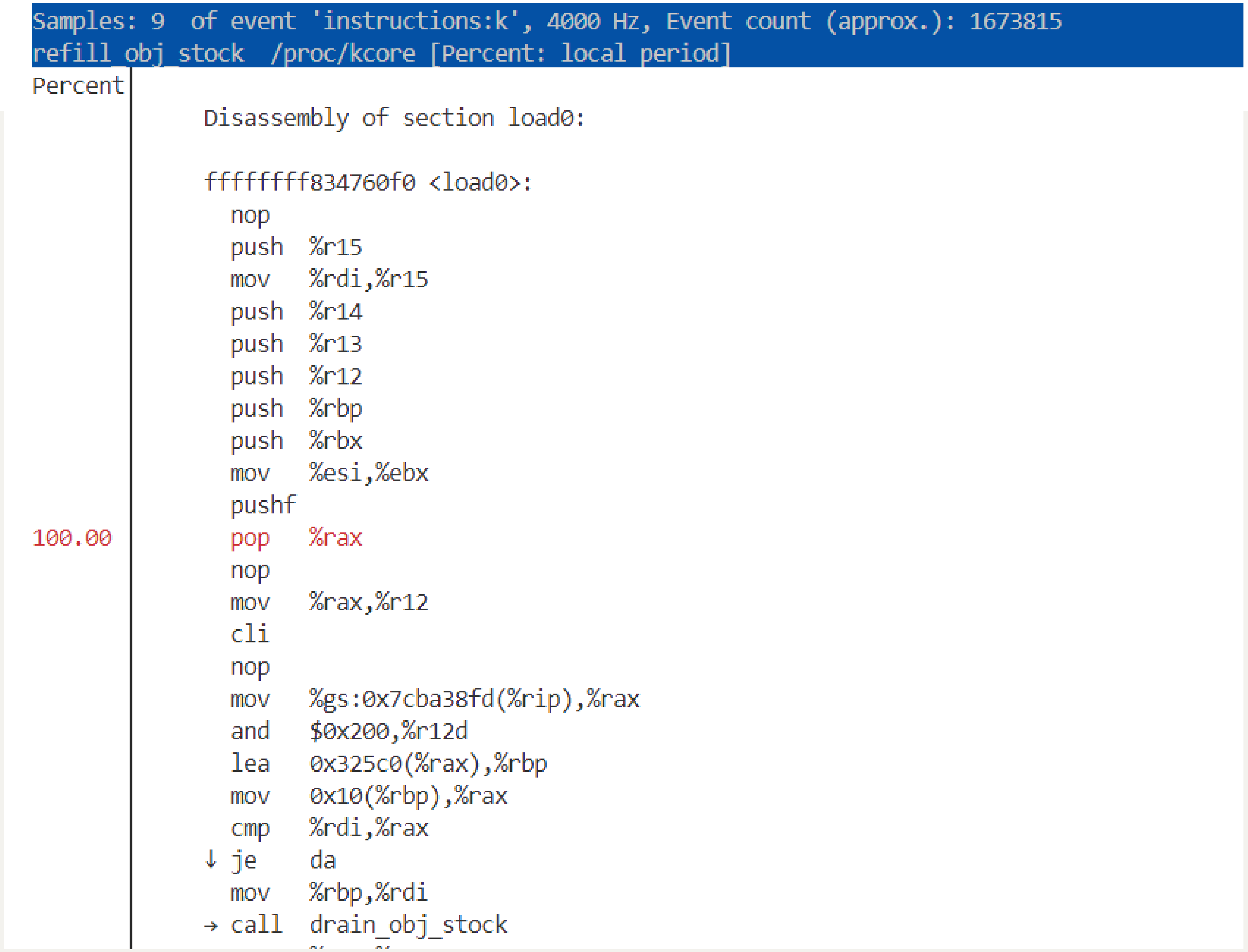
Sample Code for Perf



```
subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ gcc -g sample.c
subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ sudo perf record -e cycles,instructions:u,instructions:k,LLC-loads,LLC-load-misses,branch-mis
ses,bus-cycles,alignment-faults,page-faults ./a.out
x = 100000
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.026 MB perf.data (65
```

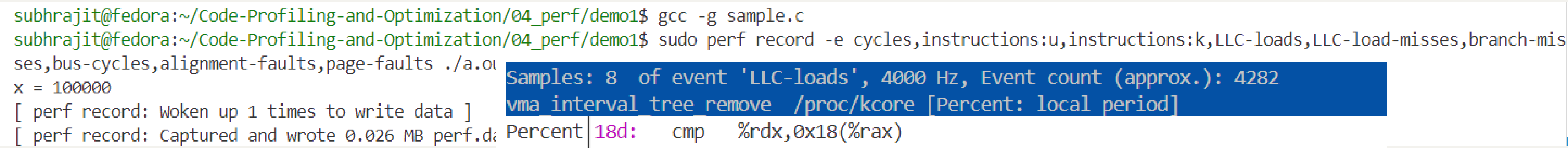


```c
#include <stdio.h>

int sample_function(int p)
{
    int n = 0;
    for (int i = 0; i < p; i++)
    {
        n = n + 1;
    }
    return n;
}

int main()
{
    int x = sample_function(100000);
    printf("x = %d\n", x);

    return 0;
}
```

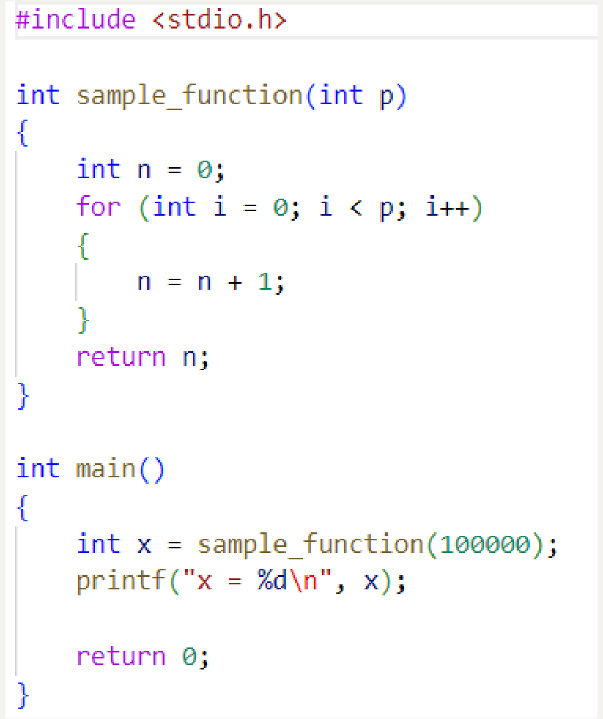


```
Samples: 9  of event 'instructions:k', 4000 Hz, Event count (approx.): 1673815
refill_obj_stock  /proc/kcore [Percent: local period]
Percent│
       │
       │     Disassembly of section load0:
       │
       │     ffffffff834760f0 <load0>:
       │       nop
       │       push   %r15
       │       mov    %rdi,%r15
       │       push   %r14
       │       push   %r13
       │       push   %r12
       │       push   %rbp
       │       push   %rbx
       │       mov    %esi,%ebx
       │       pushf
100.00 │       pop    %rax
       │       nop
       │       mov    %rax,%r12
       │       cli
       │       nop
       │       mov    %gs:0x7cba38fd(%rip),%rax
       │       and    $0x200,%r12d
       │       lea    0x325c0(%rax),%rbp
       │       mov    0x10(%rbp),%rax
       │       cmp    %rdi,%rax
       │     ↓ je     da
       │       mov    %rbp,%rdi
       │     → call   drain_obj_stock
```

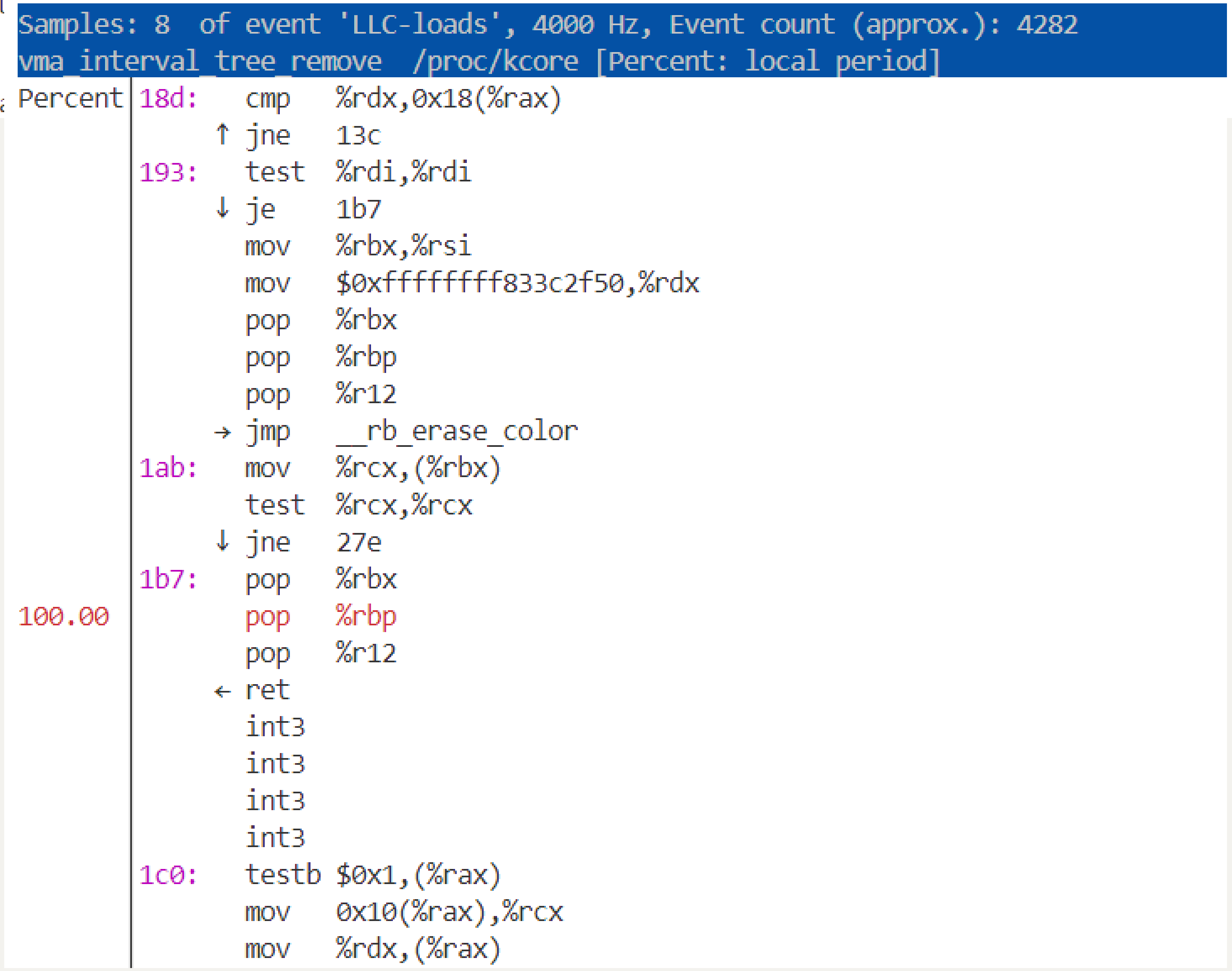
Sample Code for Perf



```
subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ gcc -g sample.c
subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ sudo perf record -e cycles,instructions:u,instructions:k,LLC-loads,LLC-load-misses,branch-mis
ses,bus-cycles,alignment-faults,page-faults ./a.ou
x = 100000
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.026 MB perf.da
```

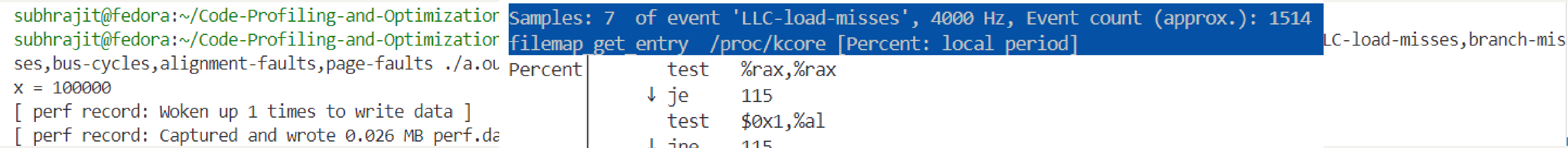


```c
#include <stdio.h>

int sample_function(int p)
{
    int n = 0;
    for (int i = 0; i < p; i++)
    {
        n = n + 1;
    }
    return n;
}

int main()
{
    int x = sample_function(100000);
    printf("x = %d\n", x);

    return 0;
}
```

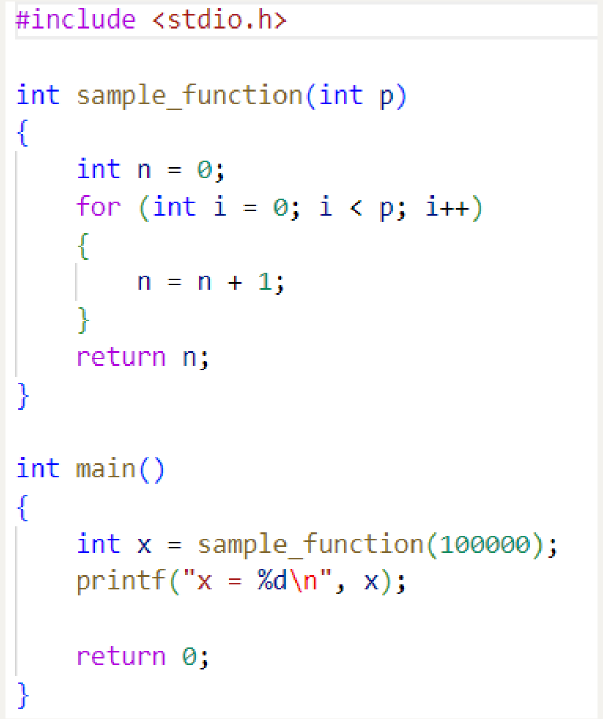


```
Samples: 8  of event 'LLC-loads', 4000 Hz, Event count (approx.): 4282
vma_interval_tree_remove  /proc/kcore [Percent: local period]
Percent│18d:    cmp    %rdx,0x18(%rax)
       │      ↑ jne    13c
       │193:    test   %rdi,%rdi
       │      ↓ je     1b7
       │        mov    %rbx,%rsi
       │        mov    $0xffffffff833c2f50,%rdx
       │        pop    %rbx
       │        pop    %rbp
       │        pop    %r12
       │      → jmp    __rb_erase_color
       │1ab:    mov    %rcx,(%rbx)
       │        test   %rcx,%rcx
       │      ↓ jne    27e
       │1b7:    pop    %rbx
100.00 │        pop    %rbp
       │        pop    %r12
       │      ← ret
       │        int3
       │        int3
       │        int3
       │        int3
       │1c0:    testb  $0x1,(%rax)
       │        mov    0x10(%rax),%rcx
       │        mov    %rdx,(%rax)
```

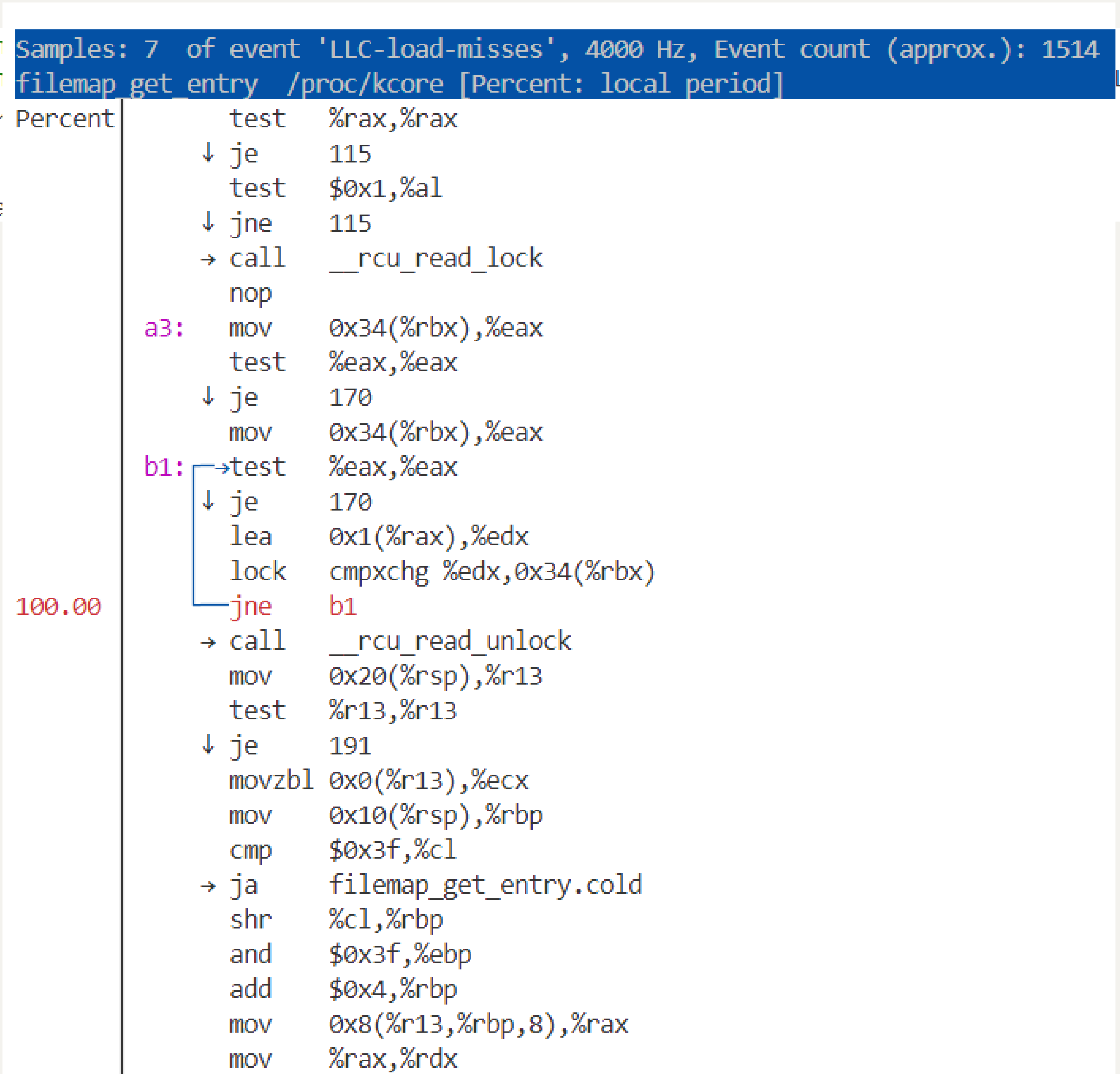
Sample Code for Perf



```
subhrajit@fedora:~/Code-Profiling-and-Optimizatior
subhrajit@fedora:~/Code-Profiling-and-Optimizatior                                     LC-load-misses,branch-mis
ses,bus-cycles,alignment-faults,page-faults ./a.ou
x = 100000
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.026 MB perf.da
```

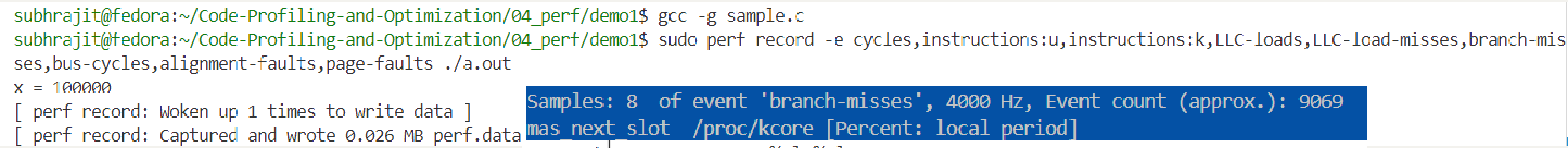


```c
#include <stdio.h>

int sample_function(int p)
{
    int n = 0;
    for (int i = 0; i < p; i++)
    {
        n = n + 1;
    }
    return n;
}

int main()
{
    int x = sample_function(100000);
    printf("x = %d\n", x);

    return 0;
}
```

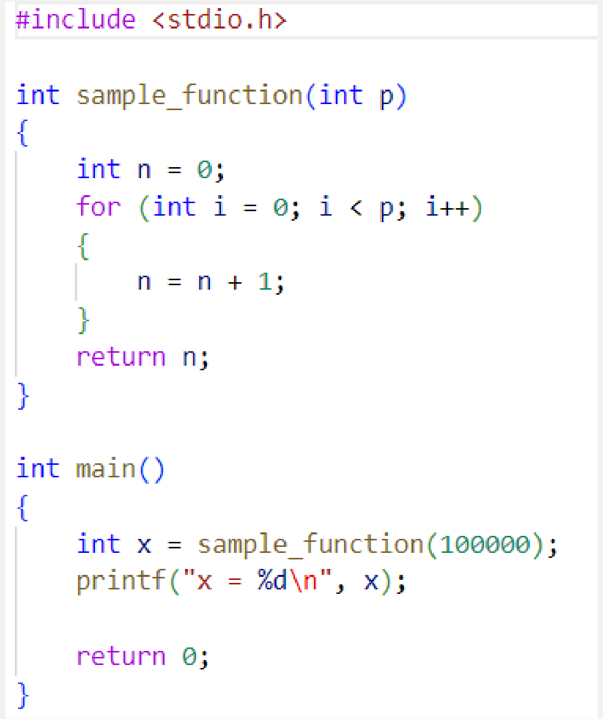


```
Samples: 7  of event 'LLC-load-misses', 4000 Hz, Event count (approx.): 1514
filemap_get_entry  /proc/kcore [Percent: local period]
Percent │        test   %rax,%rax
        │      ↓ je     115
        │        test   $0x1,%al
        │      ↓ jne    115
        │      → call   __rcu_read_lock
        │        nop
        │a3:     mov    0x34(%rbx),%eax
        │        test   %eax,%eax
        │      ↓ je     170
        │        mov    0x34(%rbx),%eax
        │b1:  ┌─→test   %eax,%eax
        │     │↓ je     170
        │     │  lea    0x1(%rax),%edx
        │     │  lock   cmpxchg %edx,0x34(%rbx)
 100.00 │     └──jne    b1
        │      → call   __rcu_read_unlock
        │        mov    0x20(%rsp),%r13
        │        test   %r13,%r13
        │      ↓ je     191
        │        movzbl 0x0(%r13),%ecx
        │        mov    0x10(%rsp),%rbp
        │        cmp    $0x3f,%cl
        │      → ja     filemap_get_entry.cold
        │        shr    %cl,%rbp
        │        and    $0x3f,%ebp
        │        add    $0x4,%rbp
        │        mov    0x8(%r13,%rbp,8),%rax
        │        mov    %rax,%rdx
```

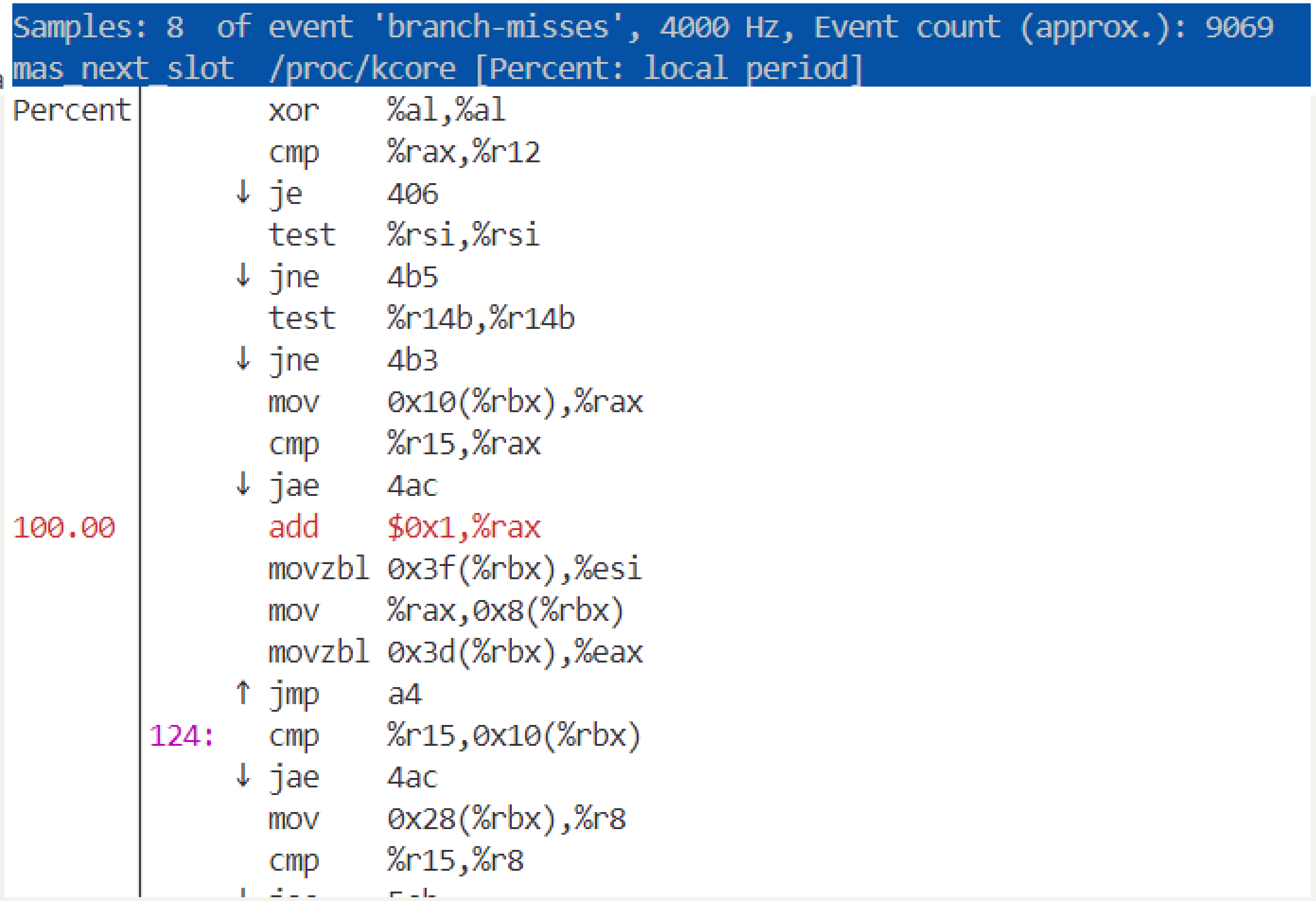
Sample Code for Perf



```
subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ gcc -g sample.c
subhrajit@fedora:~/Code-Profiling-and-Optimization/04_perf/demo1$ sudo perf record -e cycles,instructions:u,instructions:k,LLC-loads,LLC-load-misses,branch-mis
ses,bus-cycles,alignment-faults,page-faults ./a.out
x = 100000
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.026 MB perf.data
```

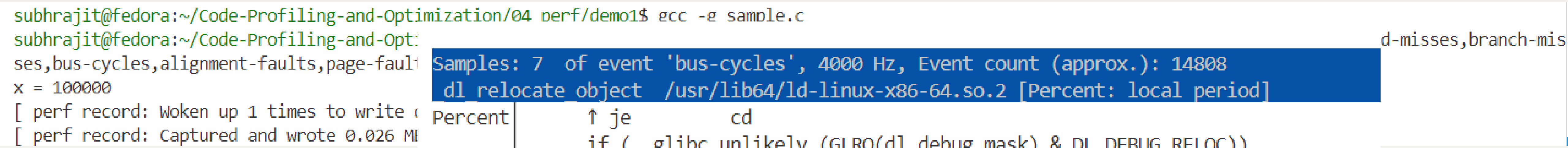


```c
#include <stdio.h>

int sample_function(int p)
{
    int n = 0;
    for (int i = 0; i < p; i++)
    {
        n = n + 1;
    }
    return n;
}

int main()
{
    int x = sample_function(100000);
    printf("x = %d\n", x);

    return 0;
}
```

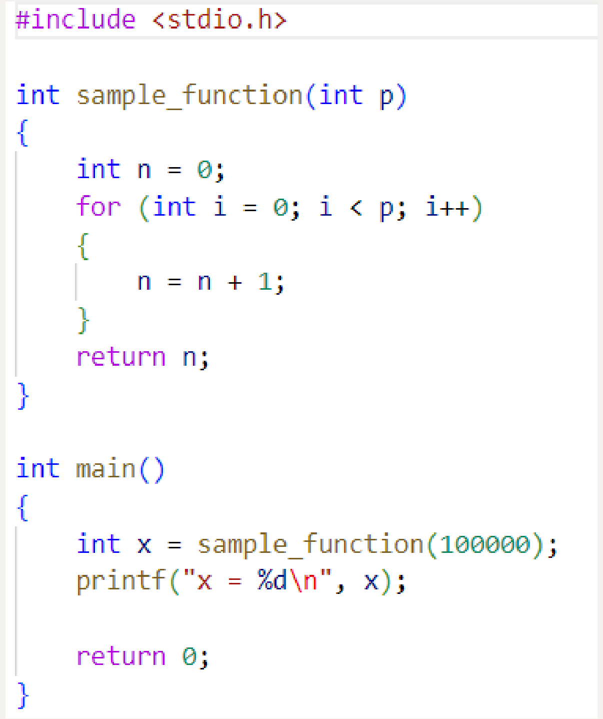


```
Samples: 8  of event 'branch-misses', 4000 Hz, Event count (approx.): 9069
mas_next_slot  /proc/kcore [Percent: local period]
Percent│         xor    %al,%al
       │         cmp    %rax,%r12
       │       ↓ je     406
       │         test   %rsi,%rsi
       │       ↓ jne    4b5
       │         test   %r14b,%r14b
       │       ↓ jne    4b3
       │         mov    0x10(%rbx),%rax
       │         cmp    %r15,%rax
       │       ↓ jae    4ac
100.00 │         add    $0x1,%rax
       │         movzbl 0x3f(%rbx),%esi
       │         mov    %rax,0x8(%rbx)
       │         movzbl 0x3d(%rbx),%eax
       │       ↑ jmp    a4
       │124:     cmp    %r15,0x10(%rbx)
       │       ↓ jae    4ac
       │         mov    0x28(%rbx),%r8
       │         cmp    %r15,%r8
```

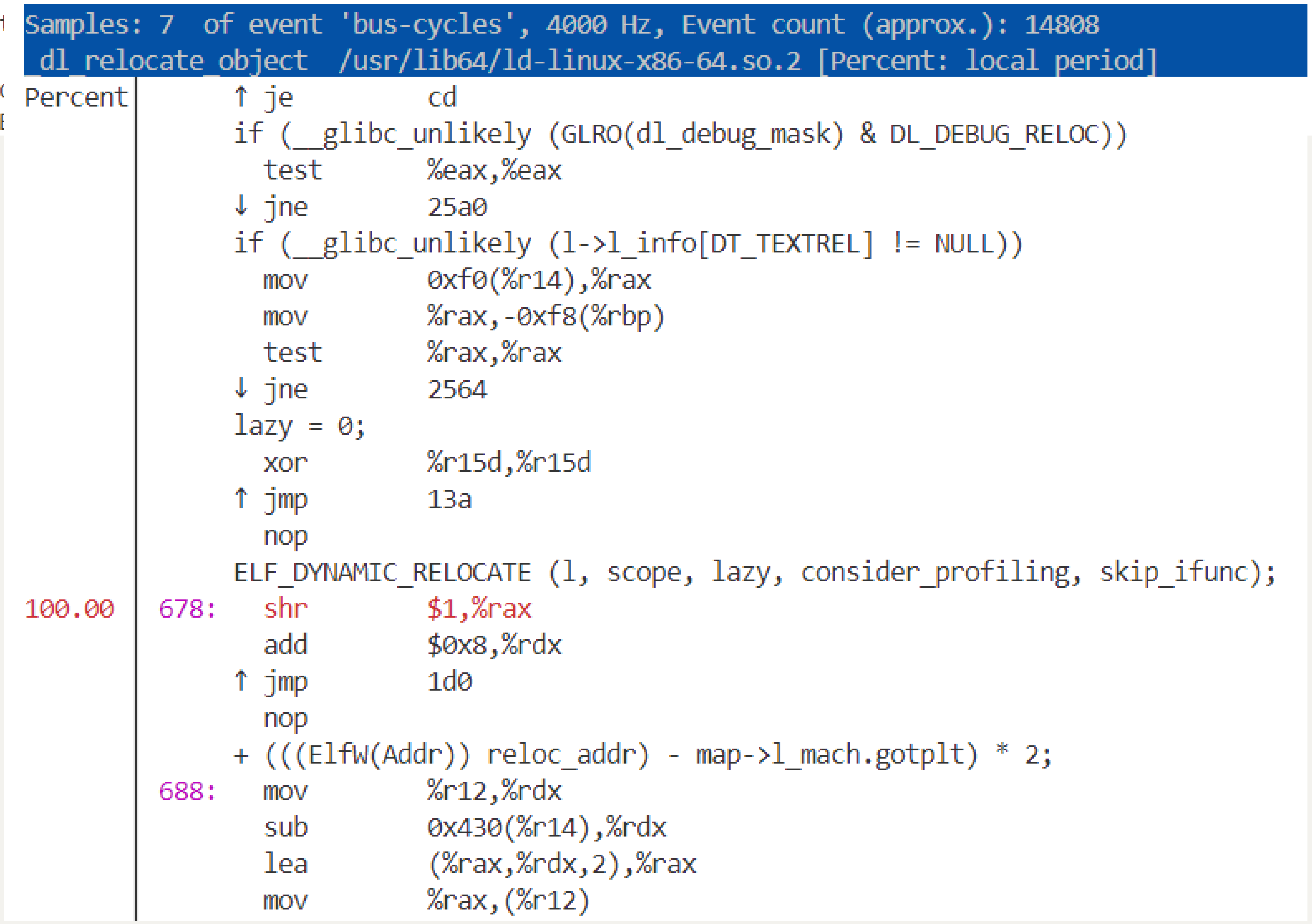
Sample Code for Perf



```
subhrajit@fedora:~/Code-Profiling-and-Optimization/04 perf/demo1$ gcc -g sample.c
subhrajit@fedora:~/Code-Profiling-and-Opt                                          d-misses,branch-mis
ses,bus-cycles,alignment-faults,page-fault
x = 100000
[ perf record: Woken up 1 times to write 
[ perf record: Captured and wrote 0.026 MB
```

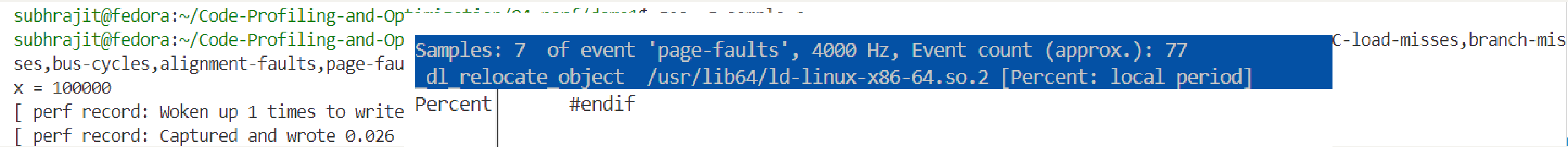


```c
#include <stdio.h>

int sample_function(int p)
{
    int n = 0;
    for (int i = 0; i < p; i++)
    {
        n = n + 1;
    }
    return n;
}

int main()
{
    int x = sample_function(100000);
    printf("x = %d\n", x);

    return 0;
}
```

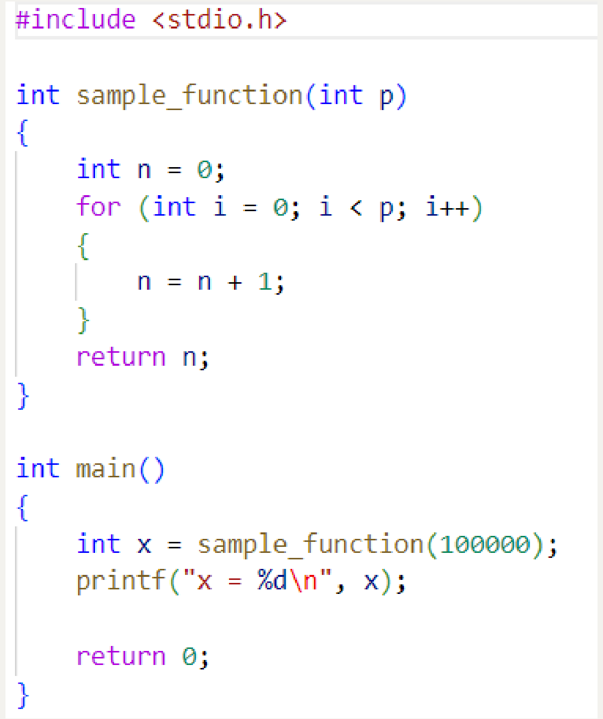


```
Samples: 7  of event 'bus-cycles', 4000 Hz, Event count (approx.): 14808
 dl_relocate_object  /usr/lib64/ld-linux-x86-64.so.2 [Percent: local period]
Percent│      ↑ je         cd
       │      if (__glibc_unlikely (GLRO(dl_debug_mask) & DL_DEBUG_RELOC))
       │        test       %eax,%eax
       │      ↓ jne        25a0
       │      if (__glibc_unlikely (l->l_info[DT_TEXTREL] != NULL))
       │        mov        0xf0(%r14),%rax
       │        mov        %rax,-0xf8(%rbp)
       │        test       %rax,%rax
       │      ↓ jne        2564
       │      lazy = 0;
       │        xor        %r15d,%r15d
       │      ↑ jmp        13a
       │        nop
       │      ELF_DYNAMIC_RELOCATE (l, scope, lazy, consider_profiling, skip_ifunc);
100.00 │ 678:   shr        $1,%rax
       │        add        $0x8,%rdx
       │      ↑ jmp        1d0
       │        nop
       │      + (((ElfW(Addr)) reloc_addr) - map->l_mach.gotplt) * 2;
       │ 688:   mov        %r12,%rdx
       │        sub        0x430(%r14),%rdx
       │        lea        (%rax,%rdx,2),%rax
       │        mov        %rax,(%r12)
```

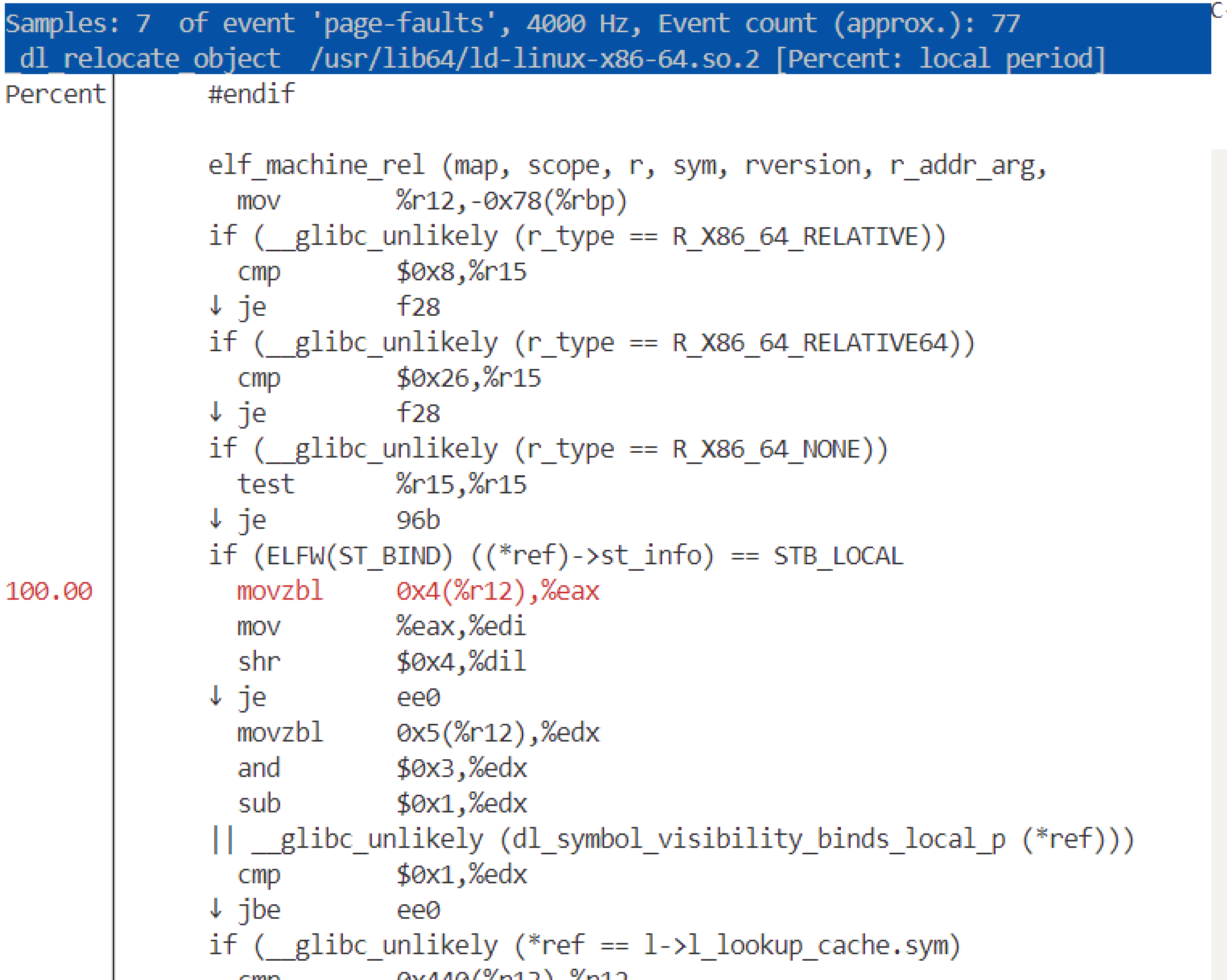
Sample Code for Perf



```
subhrajit@fedora:~/Code-Profiling-and-Op
subhrajit@fedora:~/Code-Profiling-and-Op                                    C-load-misses,branch-mis
ses,bus-cycles,alignment-faults,page-fau
x = 100000
[ perf record: Woken up 1 times to write
[ perf record: Captured and wrote 0.026
```



```c
#include <stdio.h>

int sample_function(int p)
{
    int n = 0;
    for (int i = 0; i < p; i++)
    {
        n = n + 1;
    }
    return n;
}

int main()
{
    int x = sample_function(100000);
    printf("x = %d\n", x);

    return 0;
}
```



```
Samples: 7  of event 'page-faults', 4000 Hz, Event count (approx.): 77
 dl_relocate_object  /usr/lib64/ld-linux-x86-64.so.2 [Percent: local period]
Percent│      #endif
       │
       │      elf_machine_rel (map, scope, r, sym, rversion, r_addr_arg,
       │        mov      %r12,-0x78(%rbp)
       │      if (__glibc_unlikely (r_type == R_X86_64_RELATIVE))
       │        cmp      $0x8,%r15
       │      ↓ je       f28
       │      if (__glibc_unlikely (r_type == R_X86_64_RELATIVE64))
       │        cmp      $0x26,%r15
       │      ↓ je       f28
       │      if (__glibc_unlikely (r_type == R_X86_64_NONE))
       │        test     %r15,%r15
       │      ↓ je       96b
       │      if (ELFW(ST_BIND) ((*ref)->st_info) == STB_LOCAL
100.00 │        movzbl   0x4(%r12),%eax
       │        mov      %eax,%edi
       │        shr      $0x4,%dil
       │      ↓ je       ee0
       │        movzbl   0x5(%r12),%edx
       │        and      $0x3,%edx
       │        sub      $0x1,%edx
       │      || __glibc_unlikely (dl_symbol_visibility_binds_local_p (*ref)))
       │        cmp      $0x1,%edx
       │      ↓ jbe      ee0
       │      if (__glibc_unlikely (*ref == l->l_lookup_cache.sym)
       │        cmp      0x440(%r12),%r12
```

★ perf cheat sheet ★

JULIA EVANS
@b0rk

sourced from brendangregg.com/perf.html, which has <u>many</u> more great examples

important command line arguments

- -a : entire system
- -g : record stack traces
- -e : choose an <u>event</u> to record
- -p : specify a PID
- -F : pick sample frequency

perf top: get updates live!

sampling

```
# Sample CPUs at 49 Hertz, show top symbols:
perf top -F 49

# Sample CPUs, show top process names and segments:
perf top -ns comm,dso
```

tracing

```
# Count system calls by process, refreshing every 1 second:
perf top -e raw_syscalls:sys_enter -ns comm -d 1

# Count sent network packets by process, rolling output:
stdbuf -oL perf top -e net:net_dev_xmit -ns comm | strings
```

perf stat : count events ! CPU counters !

```
# CPU counter statistics for COMMAND:
perf stat COMMAND

# *Detailed* CPU counter statistics for COMMAND:
perf stat -ddd command

# Various basic CPU statistics, system wide:
perf stat -e cycles,instructions,cache-misses -a

# Count system calls for PID, until Ctrl-C:
perf stat -e 'syscalls:sys_enter_*' -p PID

# Count block device I/O events for the entire system, for 10 seconds:
perf stat -e 'block:*' -a sleep 10
```

Reporting

```
# Show perf.data in an ncurses browser:
perf report

# Show perf.data as a text report:
perf report --stdio

# List all events from perf.data:
perf script

# Annotate assembly instructions from perf.data
# with percentages
perf annotate [--stdio]
```

perf trace : trace system calls & other events

```
# Trace syscalls system-wide
perf trace

# Trace syscalls for PID
perf trace -p PID
```

perf record : record profiling data

records into perf.data file

```
# Sample CPU functions for COMMAND, at 99 Hertz:
perf record -F 99 COMMAND

# Sample CPU functions for PID, until Ctrl-C:
perf record -p PID

# Sample CPU functions for PID, for 10 seconds:
perf record -p PID sleep 10

# Sample CPU stack traces for PID, for 10 seconds:
perf record -p PID -g -- sleep 10

# Sample CPU stack traces for PID, using DWARF to unwind stack:
perf record -p PID --call-graph dwarf
```

perf record : record tracing data

records into perf.data file

```
# Trace new processes, until Ctrl-C:
perf record -e sched:sched_process_exec -a

# Trace all context-switches, until Ctrl-C:
perf record -e context-switches -a

# Trace all context-switches with stack traces, for 10 seconds:
perf record -e context-switches -ag -- sleep 10

# Trace all page faults with stack traces, until Ctrl-C:
perf record -e page-faults -ag
```

adding new trace events

```
# Add a tracepoint for kernel function tcp_sendmsg():
perf probe 'tcp_sendmsg'

# Trace previously created probe:
perf record -e -a probe:tcp_sendmsg

# Add a tracepoint for myfunc() return, and include the retval as a string:
perf probe 'myfunc%return +0($retval):string'
```

these need kernel debuginfo

```
# Trace previous probe when size > 0, and state is not TCP_ESTABLISHED(1):
perf record -e -a probe:tcp_sendmsg --filter 'size > 0 && skc_state != 1' -a

# Add a tracepoint for do_sys_open() with the filename as a string:
perf probe 'do_sys_open filename:string'
```

perf_event_open()

- https://www.man7.org/linux/man-pages/man2/perf_event_open.2.html