# ΕΛΠ 605: Προχωρημένη Αρχιτεκτονική Υπολογιστών

## Εργαστήριο Αρ. 4

perf Linux Performance Tool

# Linux Performance Monitoring Tools

ps

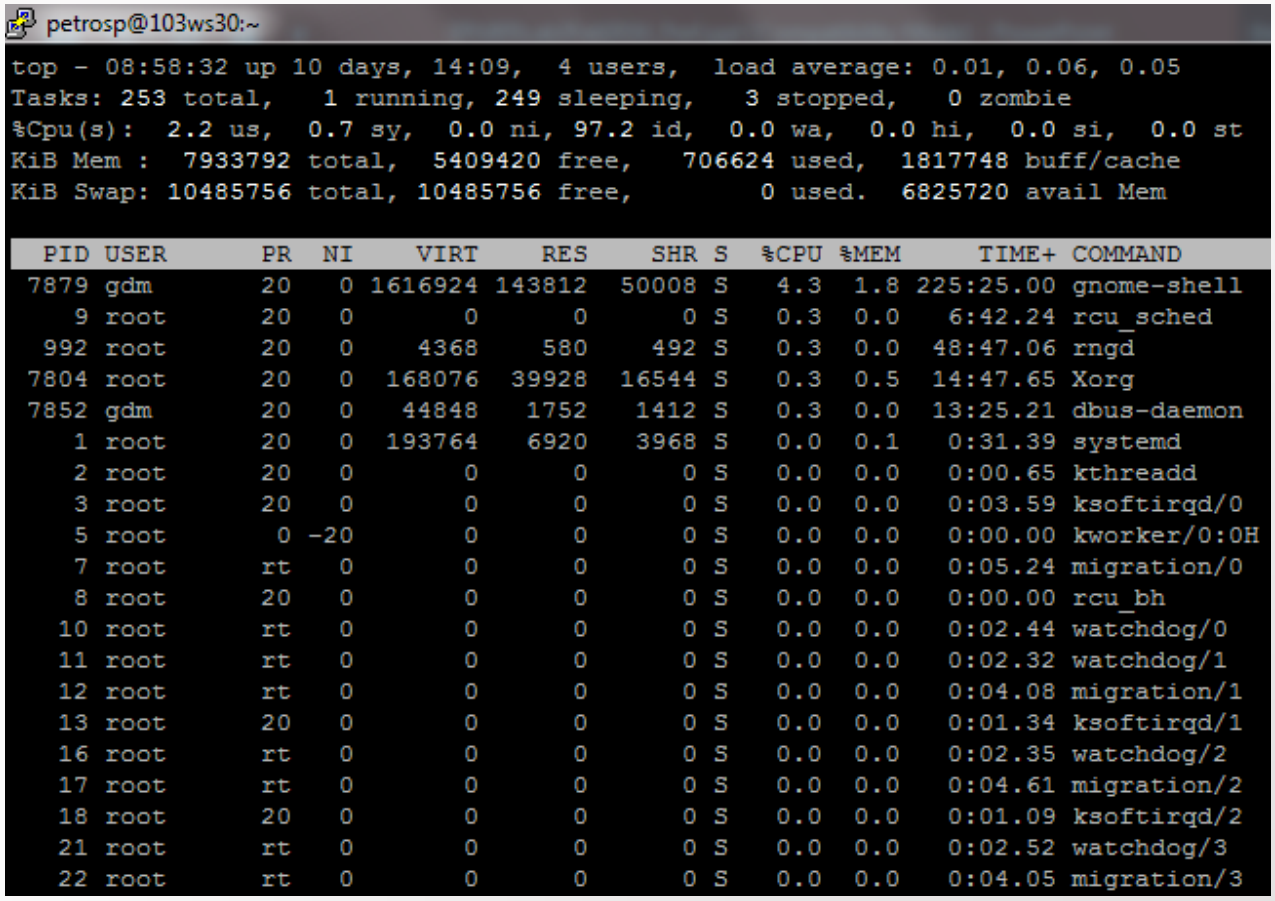
top

Press (Shift+P)

to sort processes

as per CPU

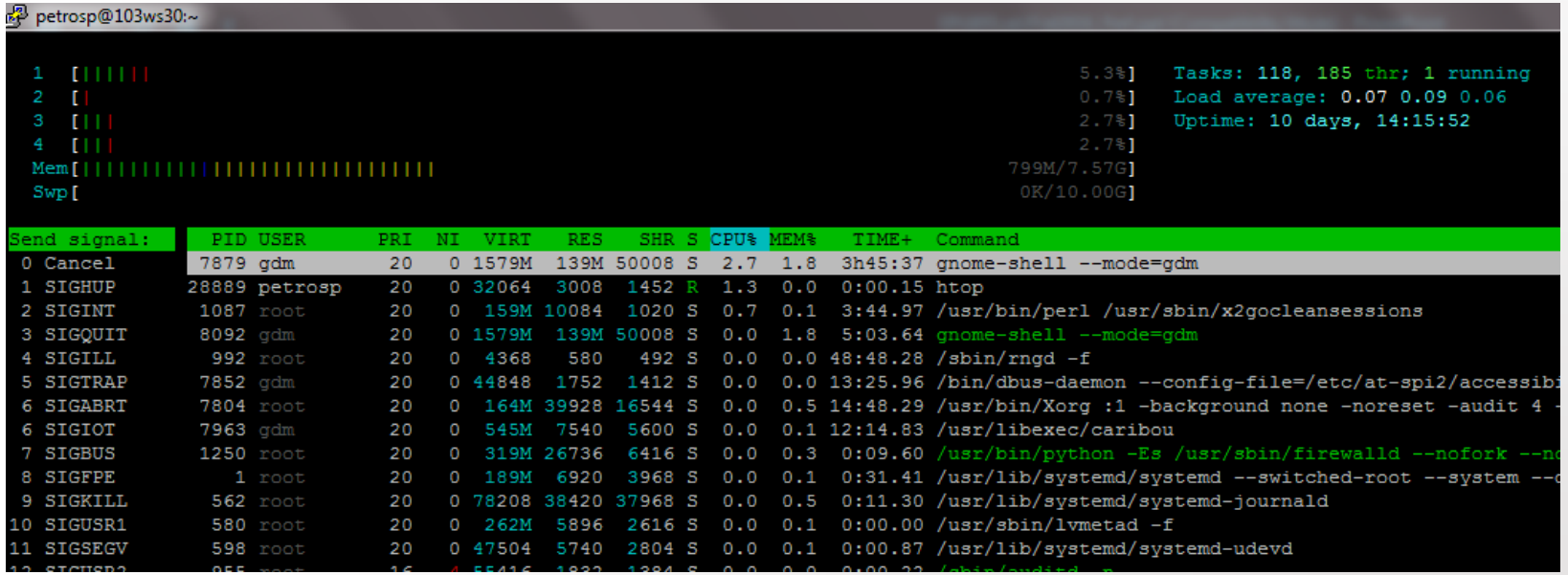
utilization



# Linux Performance Monitoring Tools

Htop (https://linux.die.net/man/1/htop)

an interactive system-monitor process-viewer



# ***perf***: Linux ***profiling*** with performance counters

***Performance counters*** are CPU hardware registers that count hardware events such as instructions executed, cache-misses suffered, or branches mispredicted.

***perf*** provides rich generalized abstractions over hardware specific capabilities. Among others, it provides per task, per CPU and per-workload counters, sampling on top of these and source code event annotation. Perf gives you visibility where the Hotspots of your program are.

https://perf.wiki.kernel.org/index.php/Main_Page

# Intel Core Performance Monitor Unit (PMU)

The core PMU's capability is similar to those described in Section 18.7.1 and Section 18.8, with some differences and enhancements relative to Intel microarchitecture code name Westmere summarized in Table 18-25.

Table 18-25. Core PMU Comparison

| Box | Intel® microarchitecture code name Sandy Bridge | Intel® microarchitecture code name Westmere | Comment |
|---|---|---|---|
| # of Fixed counters per thread | 3 | 3 | Use CPUID to enumerate # of counters. |
| # of general-purpose counters per core | 8 | 8 | |
| Counter width (R,W) | R:48 , W: 32/48 | R:48, W:32 | See Section 18.2.2.3. |
| # of programmable counters per thread | 4 or (8 if a core not shared by two threads) | 4 | Use CPUID to enumerate # of counters. |
| Precise Event Based Sampling (PEBS) Events | See Table 18-27 | See Table 18-10 | IA32_PMC4-IA32_PMC7 do not support PEBS. |
| PEBS-Load Latency | See Section 18.9.4.2;<br>Data source encoding,<br>STLB miss encoding,<br>Lock transaction encoding | Data source encoding | |
| PEBS-Precise Store | Section 18.9.4.3 | No | |
| PEBS-PDIR | yes (using precise INST_RETIRED.ALL) | No | |
| Off-core Response Event | MSR 1A6H and 1A7H; Extended request and response types | MSR 1A6H and 1A7H, limited response types | Nehalem supports 1A6H only. |

# Basic perf Options

Dynamic Optimization using the Performance Counters for CPU and Memory Profiling.

```
>perf list
List of pre-defined events (to be used in -e):
  cpu-cycles OR cycles                                [Hardware event]
  instructions                                        [Hardware event]
  cache-references                                    [Hardware event]
  cache-misses                                        [Hardware event]
  branch-instructions OR branches                     [Hardware event]
  branch-misses                                       [Hardware event]
  bus-cycles                                          [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend     [Hardware event]
  stalled-cycles-backend OR idle-cycles-backend       [Hardware event]
  ref-cycles                                          [Hardware event]

  cpu-clock                                     [Software event]
  task-clock                                    [Software event]
  page-faults OR faults                         [Software event]
  context-switches OR cs                        [Software event]
  cpu-migrations OR migrations                  [Software event]
  minor-faults                                  [Software event]
  major-faults                                  [Software event]
  alignment-faults                              [Software event]
  emulation-faults                              [Software event]

  L1-dcache-loads                               [Hardware cache event]
  L1-dcache-load-misses                         [Hardware cache event]
  L1-dcache-stores                              [Hardware cache event]
  L1-dcache-store-misses                        [Hardware cache event]
  L1-dcache-prefetches                          [Hardware cache event]
```

. . .

https://perf.wiki.kernel.org/index.php/Tutorial

http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

# Perf Linux Performance Tool

```
>gcc -O3 main.c
>time ./a.out
sum = 400000034998780787062429554585290932224.00
0.376u 0.000s 0:00.37 100.0%    0+0k 16+0io 0pf+0w
>perf stat ./a.out
sum = 400000034998780787062429554585290932224.00

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

        373.038548 task-clock                #    0.998 CPUs utilized
                39 context-switches          #    0.105 K/sec
                 0 cpu-migrations            #    0.000 K/sec
               129 page-faults               #    0.346 K/sec
     1,108,828,204 cycles                    #    2.972 GHz                     [50.12%]
   <not supported> stalled-cycles-frontend
   <not supported> stalled-cycles-backend
     2,600,740,775 instructions              #    2.35  insns per cycle         [75.10%]
       400,189,661 branches                  # 1072.784 M/sec                   [75.07%]
            11,156 branch-misses             #    0.00% of all branches         [74.87%]

       0.373782171 seconds time elapsed
```

# Perf multiple events

To measure more than one event, simply provide a comma-separated list with no space:

```
perf stat -e cycles,instructions,cache-misses [...]
```

the kernel will automatically multiplex the number of events that need to be provided.

multiplexing and scaling events

If there are more events than counters, the kernel uses time multiplexing (switch frequency = HZ, generally 100 or 1000) to give each event a chance to access the monitoring hardware. Multiplexing only applies to PMU events. With multiplexing, an event is not measured all the time. At the end of the run, the tool scales the count based on total time enabled vs time running.

The actual formula is:

```
final_count = raw_count * time_enabled/time_running
```

Events are currently managed in round-robin fashion. Therefore each event will eventually get a chance to run. To avoid scaling (in the presence of only one active perf_event user), one can try and reduce the number of events. The following table provides the number of counters for a few common processors:

| Processor | Generic counters | Fixed counters |
|---|---|---|
| Intel Core | 2 | 3 |
| Intel Nehalem | 4 | 3 |

https://perf.wiki.kernel.org/index.php/Tutorial

# Perf Scaling

```
>perf stat -e cycles,cycles ./a.out
sum = 40000003499878078706242955458529093224.00

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

     1,232,324,077 cycles                    #    0.000 GHz
     1,232,336,337 cycles                    #    0.000 GHz

       0.590856656 seconds time elapsed

>perf stat -e cycles,cycles,cycles,cycles ./a.out
sum = 40000003499878078706242955458529093224.00

 Performance counter stats for './a.out':
     1,204,204,939 cycles                    #    0.000 GHz                     [74.97%]
     1,205,541,066 cycles                    #    0.000 GHz                     [74.84%]
     1,205,566,744 cycles                    #    0.000 GHz                     [74.84%]
     1,196,072,400 cycles                    #    0.000 GHz                     [75.51%]

       0.557318767 seconds time elapsed
```

# Perf Linux Performance Tool

```
>perf stat -r 10 ./a.out # Run 10 Times
sum = 40000003499878078706242955458529093222 4.00
sum = 40000003499878078706242955458529093222 4.00
...

 Performance counter stats for './a.out' (10 runs):

        372.392419 task-clock                #    0.998 CPUs utilized            ( +-  0.09% )
                38 context-switches          #    0.102 K/sec                    ( +-  0.55% )
                 1 cpu-migrations            #    0.001 K/sec                    ( +- 44.72% )
               130 page-faults               #    0.348 K/sec                    ( +-  0.12% )
     1,108,840,183 cycles                    #    2.978 GHz                      ( +-  0.02% ) [50.04%]
   <not supported> stalled-cycles-frontend
   <not supported> stalled-cycles-backend
     2,601,117,799 instructions              #    2.35  insns per cycle          ( +-  0.01% ) [75.02%]
       400,349,107 branches                  # 1075.073 M/sec                    ( +-  0.01% ) [75.01%]
            10,595 branch-misses             #    0.00% of all branches          ( +-  2.17% ) [75.07%]

       0.373017983 seconds time elapsed
```

# Perf Linux Performance Tool

**USER LEVEL INSTRUCTIONS**

```
>perf stat -e instructions:u ./a.out
sum = 40000003499878078706242955458529093224.00

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

     2,600,096,449 instructions:u            #    0.00  insns per cycle

       0.375571376 seconds time elapsed
```

**KERNEL LEVEL INSTRUCTIONS**

```
>perf stat -e instructions:k ./a.out
sum = 40000003499878078706242955458529093224.00

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

         1,867,178 instructions:k            #    0.00  insns per cycle

       0.377330791 seconds time elapsed
```

# Matrix Multiplication Examples

gcc -Werror -Wall matrix_serial_ver1.c -o matrix_serial_ver1.out
>./matrix_serial_ver1.out
Elapsed Time: 6.32 Sec.

>gcc -Werror -Wall matrix_serial_ver2.c -o matrix_serial_ver2.out
>./matrix_serial_ver2.out
Elapsed Time: 5.38 Sec.

>gcc -Werror -Wall matrix_serial_ver3.c -o matrix_serial_ver3.out
>./matrix_serial_ver3.out
Elapsed Time: 4.77 Sec.

>gcc -Werror -Wall matrix_serial_ver4.c -o matrix_serial_ver4.out
./matrix_serial_ver4.out
Elapsed Time: 4.60 Sec.

# Matrix Multiplication Examples

>perf stat -e cycles -e instructions -e cache-references -e cache-misses ./matrix_serial_ver1.out
Elapsed Time: 6.35 Sec.

```
 Performance counter stats for './matrix_serial_ver1.out':
   18,929,166,306 cycles                    #    0.000 GHz                     [50.00%]
   34,062,608,328 instructions              #    1.80  insns per cycle         [75.01%]
    1,066,881,565 cache-references                                             [74.99%]
       83,905 cache-misses                  #    0.008 % of all cache refs     [75.02%]
     6.362608904 seconds time elapsed
```

>perf stat -e cycles -e instructions -e cache-references -e cache-misses ./matrix_serial_ver2.out
Elapsed Time: 5.40 Sec.

```
 Performance counter stats for './matrix_serial_ver2.out':
   16,114,935,514 cycles                    #    0.000 GHz                     [50.00%]
   34,051,559,029 instructions              #    2.11  insns per cycle         [75.01%]
      64,741,737 cache-references                                              [74.99%]
       24,675 cache-misses                  #    0.038 % of all cache refs     [75.02%]

     5.418502175 seconds time elapsed
```

# Matrix Multiplication Examples

```
>gcc -Werror -Wall matrix_serial_ver3.c -o matrix_serial_ver3.out
> stat -e cycles -e instructions -e cache-references -e cache-misses ./matrix_serial_ver3.out
Elapsed Time: 4.79 Sec.
 Performance counter stats for './matrix_serial_ver3.out':

    14,296,893,562 cycles                    #    0.000 GHz                     [50.00%]
    20,059,297,912 instructions              #    1.40  insns per cycle         [74.99%]
     1,067,327,910 cache-references                                            [75.02%]
         107,539 cache-misses                #    0.010 % of all cache refs     [74.99%]
      4.804650637 seconds time elapsed

>gcc -Werror -Wall matrix_serial_ver4.c -o matrix_serial_ver4.out
> stat -e cycles -e instructions -e cache-references -e cache-misses ./matrix_serial_ver4.out
Elapsed Time: 4.55 Sec.
 Performance counter stats for './matrix_serial_ver4.out':
    13,582,459,137 cycles                    #    0.000 GHz                     [50.01%]
    26,075,199,666 instructions              #    1.92  insns per cycle         [75.02%]
       64,323,587 cache-references                                             [74.99%]
          30,268 cache-misses                #    0.047 % of all cache refs     [75.00%]
      4.566277059 seconds time elapsed
```

# gcc Optimizations and Branch Prediction

```
gcc main.c -O0 -o main.out
>perf stat -e cycles -e instructions -e branches -e branch-misses ./main.out
sum = 400000034998780787062429554852909322 24.00
 Performance counter stats for './main.out':
     4,120,356,326 cycles                    #    0.000 GHz                     [49.99%]
     5,306,382,355 instructions              #    1.29  insns per cycle         [74.96%]
       899,775,764 branches                                                     [75.03%]
         8,742,590 branch-misses             #    0.97% of all branches         [75.03%]
      1.387120365 seconds time elapsed

>gcc main.c -O1 -o main.out
>perf stat -e cycles -e instructions -e branches -e branch-misses ./main.out
sum = 400000034998780787062429554852909322 24.00
 Performance counter stats for './main.out':

     1,950,230,142 cycles                    #    0.000 GHz                     [50.07%]
     2,782,472,831 instructions              #    1.43  insns per cycle         [75.05%]
       600,311,976 branches                                                     [75.02%]
         1,368,519 branch-misses             #    0.23% of all branches         [74.95%]
      0.662207246 seconds time elapsed
```

# gcc Optimizations and Branch Prediction

```
>gcc main.c -O2 -o main.out
>perf stat -e cycles -e instructions -e branches -e branch-misses ./main.out
sum = 400000034998780787062429554852909322 24.00
 Performance counter stats for './main.out':
    1,109,248,454 cycles                    #    0.000 GHz                     [49.96%]
    2,599,913,162 instructions              #    2.34  insns per cycle         [75.12%]
      400,489,432 branches                                                     [75.12%]
           11,973 branch-misses             #    0.00% of all branches         [74.98%]
      0.379008882 seconds time elapsed

main.c -O3 -o main.out
stat -e cycles -e instructions -e branches -e branch-misses ./main.out
sum = 400000034998780787062429554852909322 24.00
 Performance counter stats for './main.out':
    1,108,212,355 cycles                    #    0.000 GHz                     [50.09%]
    2,603,070,814 instructions              #    2.35  insns per cycle         [75.09%]
      399,947,490 branches                                                     [75.11%]
           12,176 branch-misses             #    0.00% of all branches         [74.93%]
      0.378495972 seconds time elapsed
```

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

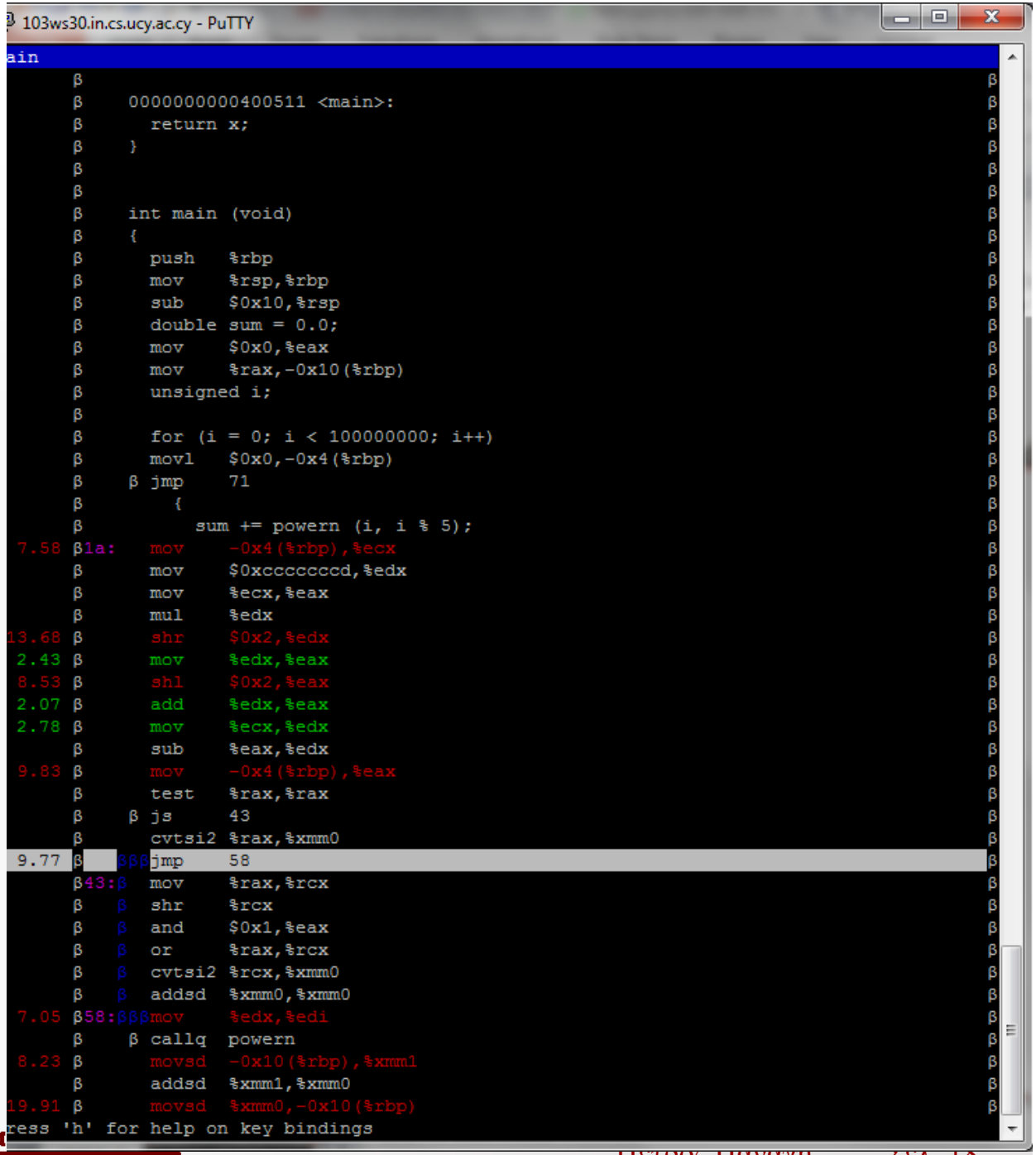
# Perf annotate

Not Working in the Labs.

perf annotate can generate sourcecode level information if the application is compiled with -ggdb.

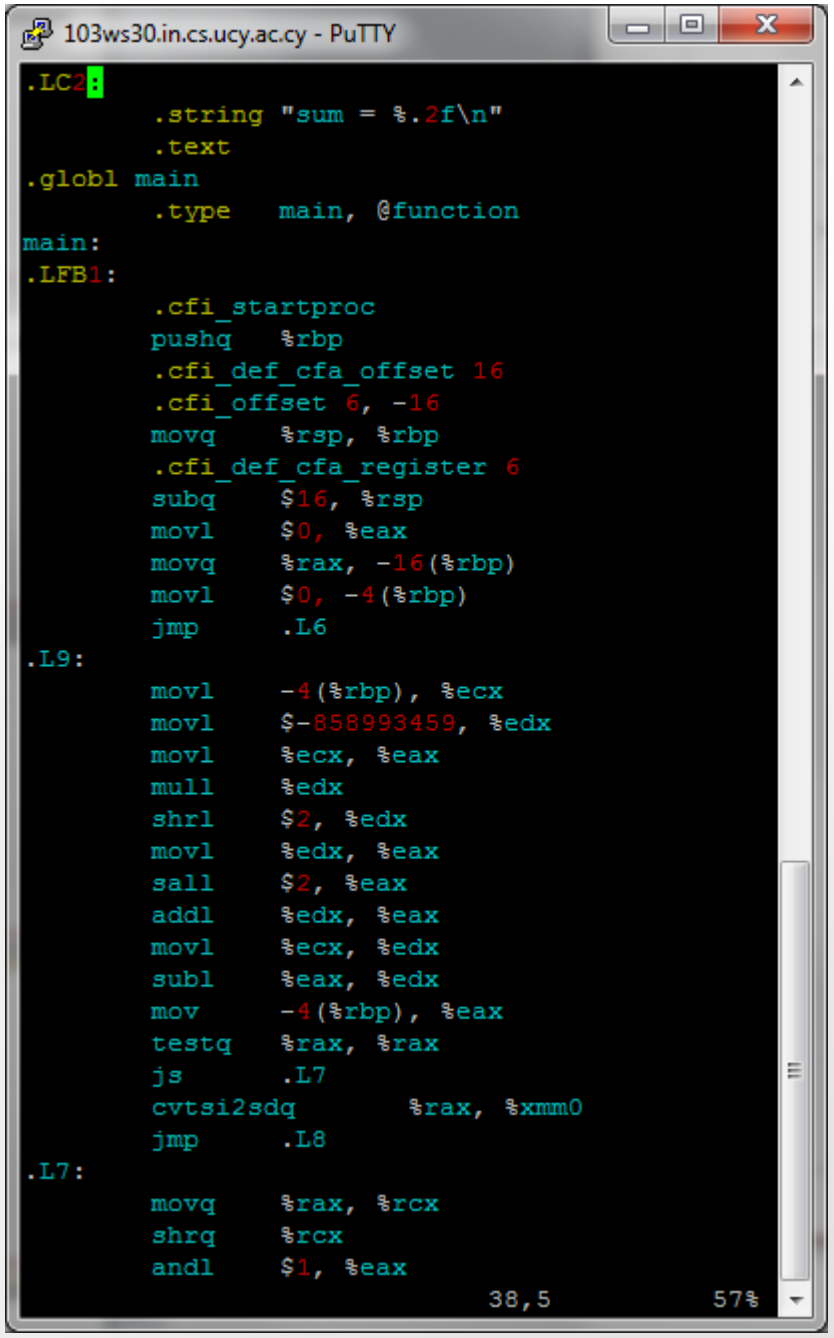
```
>ssh cs6472
>gcc -Werror -Wall -ggdb matrix_serial_ver1.c -o matrix_serial_ver1.out
> perf_3.2.0-38 record ./matrix_serial_ver1.out
WARNING: Kernel address maps (/proc/{kallsyms,modules}) are restricted,
check /proc/sys/kernel/kptr_restrict.
Samples in kernel functions may not be resolved if a suitable vmlinux
file is not found in the buildid cache or in the vmlinux path.
Samples in kernel modules won't be resolved at all.
If some relocation was applied (e.g. kexec) symbols may be misresolved
even with a suitable vmlinux or kallsyms file.

Elapsed Time: 6.40 Sec.
[ perf record: Woken up 4 times to write data ]
[ perf record: Captured and wrote 0.982 MB perf.data (~42916 samples) ]

> perf_3.2.0-38 annotate
```

# >perf annotate

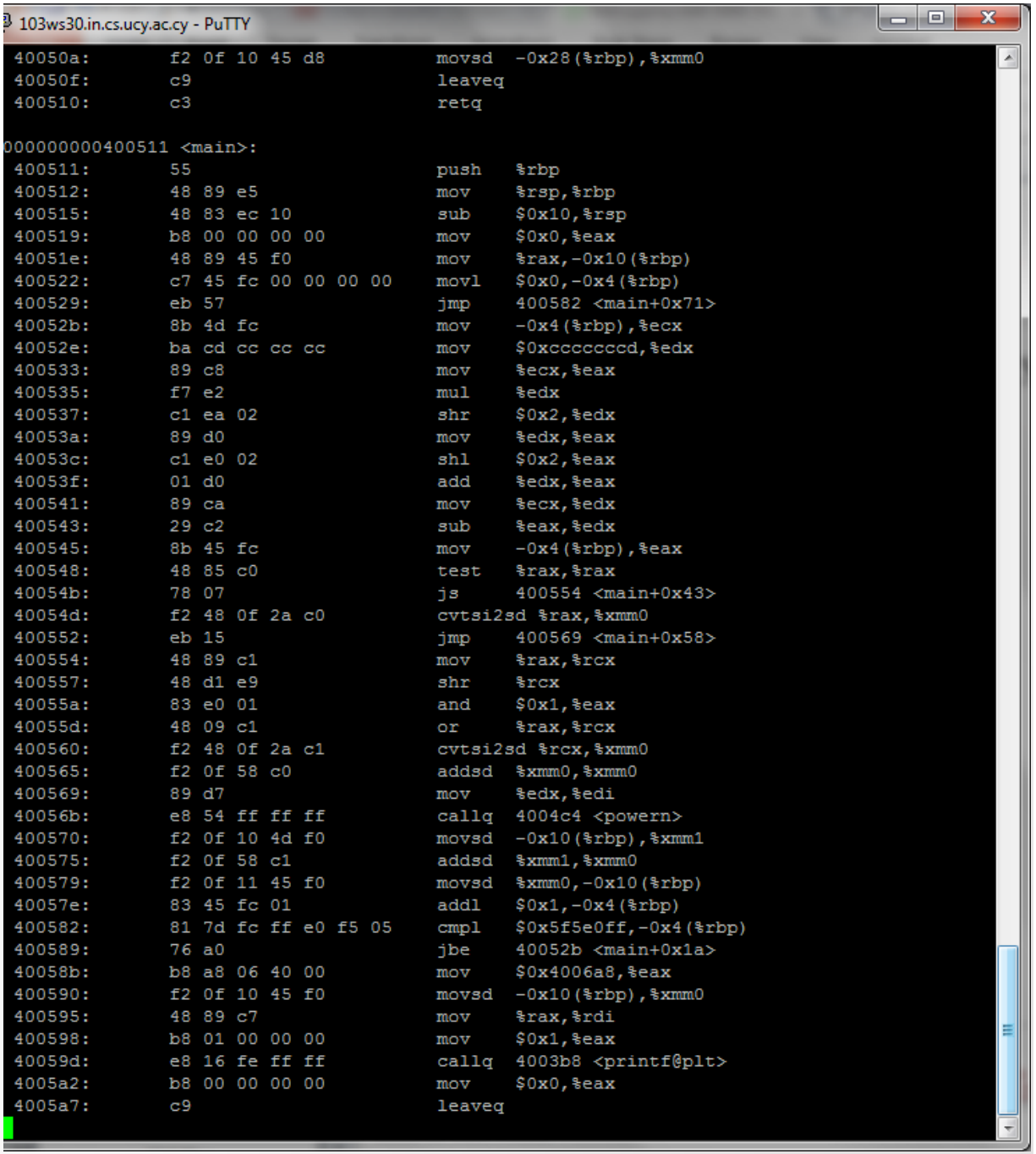


# Assembly

```
>gcc –S main.c
>vi main.s
>gcc main.s -o main.s.out
```

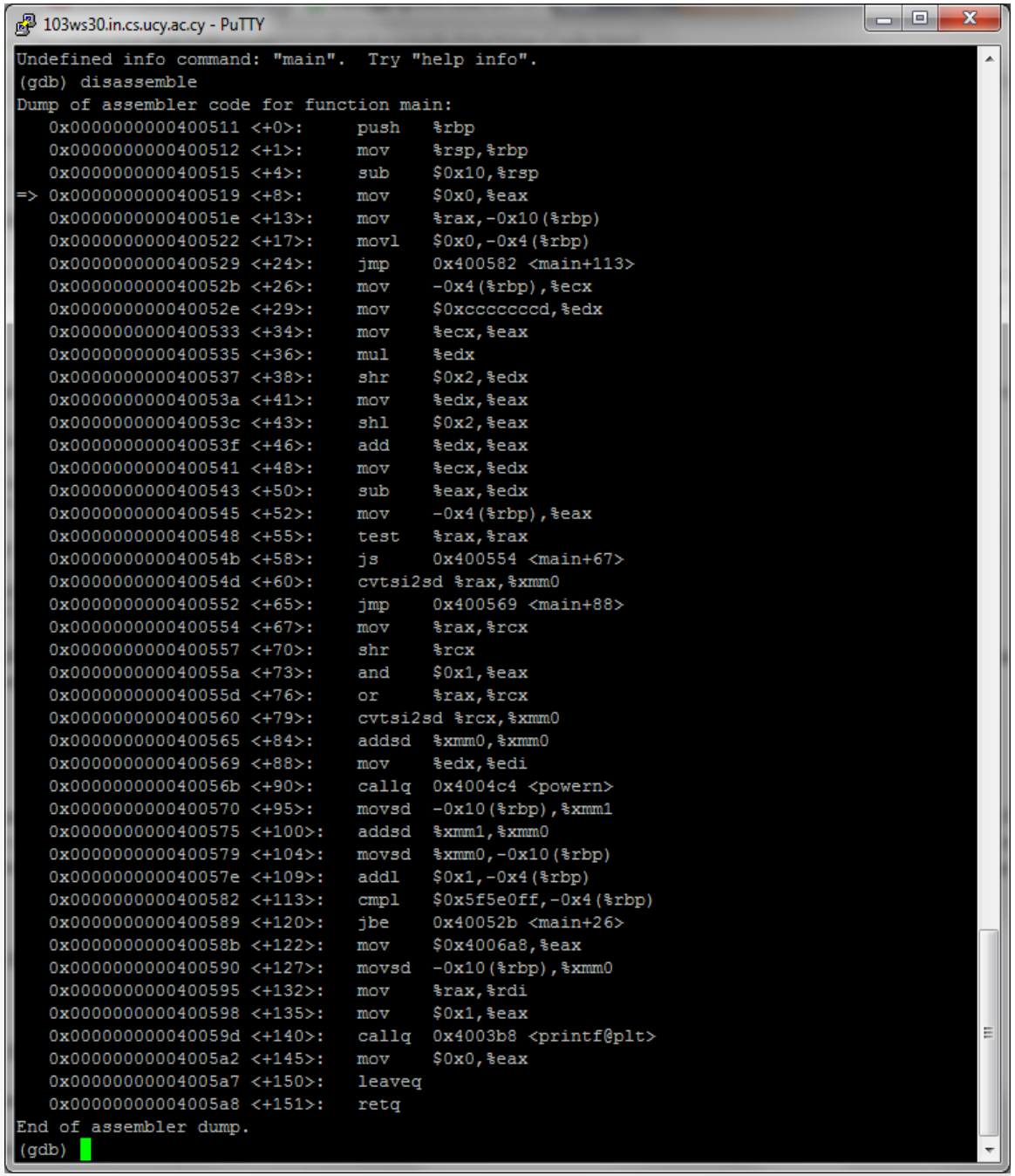


```
>objdump -d
  main.out

>hexdump -C
  main.out
```



# gdb

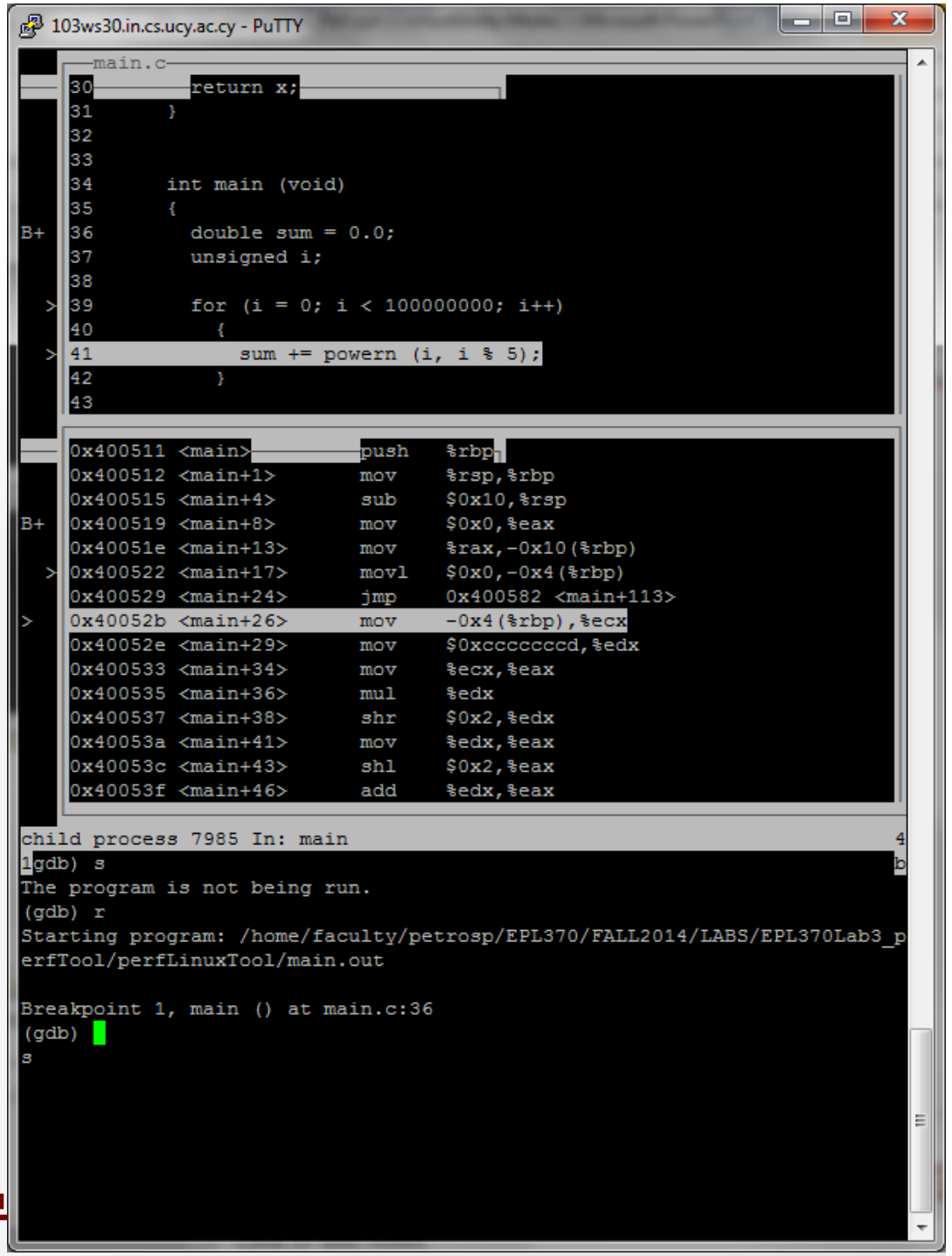
```
gcc -ggdb main.c -o main.out
bash
gdb main.out
    b main (breakpiont at main)
    r     (run)
    disassemble
    s
    s     (step)
    bt    (Backtrace)
set disassemble-next-line on
```



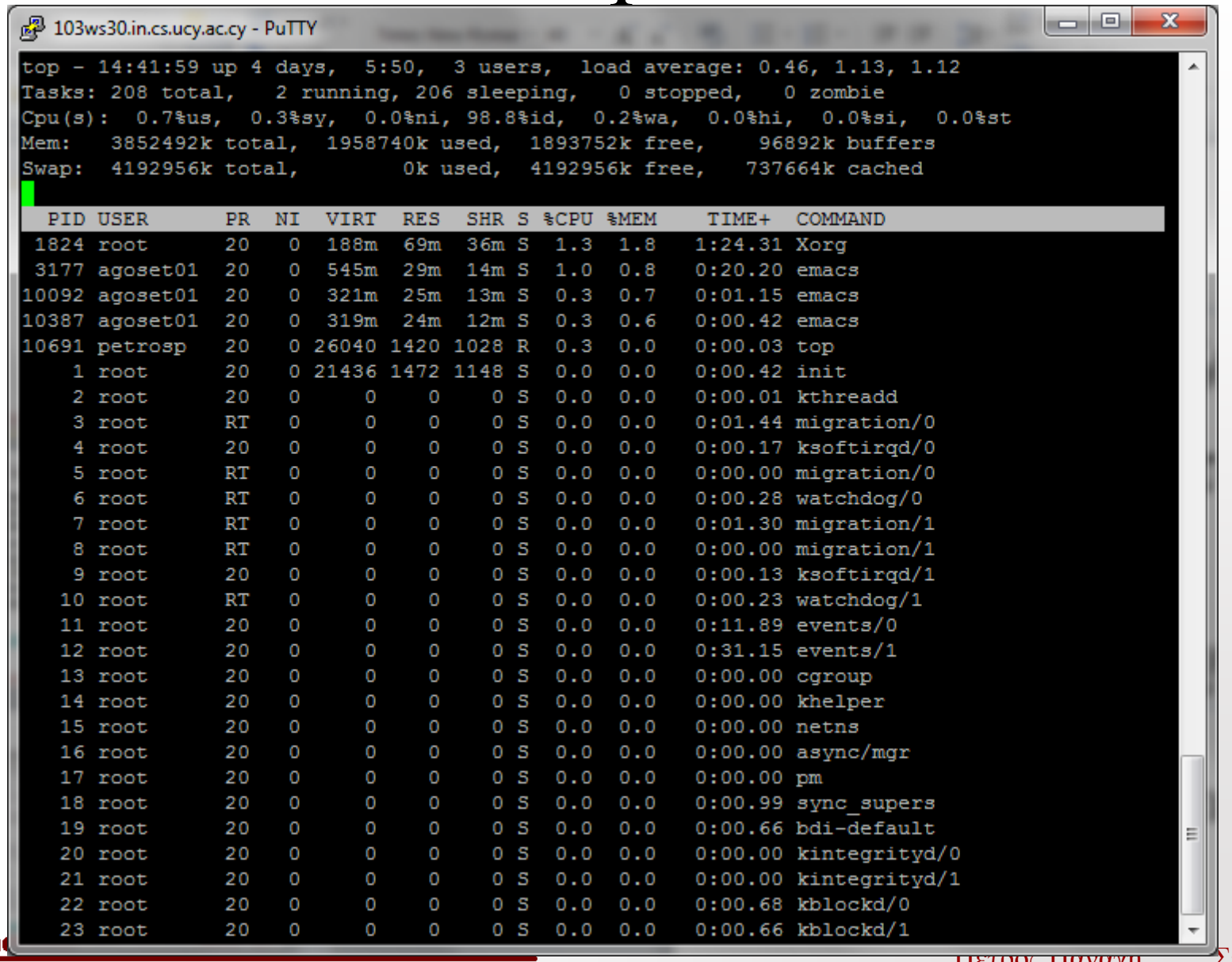
# gdb Disassemble next line

```
setenv SHELL /bin/bash
gdb a.out
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-56.el6)
..
(gdb) b main
(gdb) set disassemble-next-line on (gdb) s (gdb) r
Starting program: /home/faculty/petrosp/EPL370/FALL2012/LABS/EPL370Lab5/a.out
Breakpoint 1, main () at gprof-prog1.c:49
49          init( x );
=> 0x00000000004005bc <main+11>:       48 8d 85 60 fe ff ff      lea    -0x1a0(%rbp),%rax
   0x00000000004005c3 <main+18>:       48 89 c7                  mov    %rax,%rdi
   0x00000000004005c6 <main+21>:       e8 f9 fe ff ff            callq  0x4004c4 <init>
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.80.el6_3.4.x86_64
(gdb) s
init (x=0x7fffffffdf90) at gprof-prog1.c:8
8           for ( i = 0; i < 100; i++ ) {
=> 0x00000000004004cc <init+8>:  c7 45 fc 00 00 00 00            movl   $0x0,-0x4(%rbp)
   0x00000000004004d3 <init+15>:   eb 16              jmp    0x4004eb <init+39>
(gdb)
```

```
gcc -ggdb main.c -o main.out
bash
gdb main.out
    b main
    r
    layout asm
    ctrl-x 2
    s
```



# top


103ws30.in.cs.ucy.ac.cy - PuTTY
top - 14:41:59 up 4 days, 5:50, 3 users, load average: 0.46, 1.13, 1.12
Tasks: 208 total, 2 running, 206 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.7%us, 0.3%sy, 0.0%ni, 98.8%id, 0.2%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 3852492k total, 1958740k used, 1893752k free, 96892k buffers
Swap: 4192956k total, 0k used, 4192956k free, 737664k cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1824 root 20 0 188m 69m 36m S 1.3 1.8 1:24.31 Xorg
3177 agoset01 20 0 545m 29m 14m S 1.0 0.8 0:20.20 emacs
10092 agoset01 20 0 321m 25m 13m S 0.3 0.7 0:01.15 emacs
10387 agoset01 20 0 319m 24m 12m S 0.3 0.6 0:00.42 emacs
10691 petrosp 20 0 26040 1420 1028 R 0.3 0.0 0:00.03 top
1 root 20 0 21436 1472 1148 S 0.0 0.0 0:00.42 init
2 root 20 0 0 0 0 S 0.0 0.0 0:00.01 kthreadd
3 root RT 0 0 0 0 S 0.0 0.0 0:01.44 migration/0
4 root 20 0 0 0 0 S 0.0 0.0 0:00.17 ksoftirqd/0
5 root RT 0 0 0 0 S 0.0 0.0 0:00.00 migration/0
6 root RT 0 0 0 0 S 0.0 0.0 0:00.28 watchdog/0
7 root RT 0 0 0 0 S 0.0 0.0 0:01.30 migration/1
8 root RT 0 0 0 0 S 0.0 0.0 0:00.00 migration/1
9 root 20 0 0 0 0 S 0.0 0.0 0:00.13 ksoftirqd/1
10 root RT 0 0 0 0 S 0.0 0.0 0:00.23 watchdog/1
11 root 20 0 0 0 0 S 0.0 0.0 0:11.89 events/0
12 root 20 0 0 0 0 S 0.0 0.0 0:31.15 events/1
13 root 20 0 0 0 0 S 0.0 0.0 0:00.00 cgroup
14 root 20 0 0 0 0 S 0.0 0.0 0:00.00 khelper
15 root 20 0 0 0 0 S 0.0 0.0 0:00.00 netns
16 root 20 0 0 0 0 S 0.0 0.0 0:00.00 async/mgr
17 root 20 0 0 0 0 S 0.0 0.0 0:00.00 pm
18 root 20 0 0 0 0 S 0.0 0.0 0:00.99 sync_supers
19 root 20 0 0 0 0 S 0.0 0.0 0:00.66 bdi-default
20 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kintegrityd/0
21 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kintegrityd/1
22 root 20 0 0 0 0 S 0.0 0.0 0:00.68 kblockd/0
23 root 20 0 0 0 0 S 0.0 0.0 0:00.66 kblockd/1

# GNUPlot

http://www.gnuplot.info/
http://www.gnuplot.info/documentation.html
(log in to **cs6472** or any other machine that has gnuplot, and run **gnuplot**)

```
plot  "data.txt" using 1:2 title 'Column 2', "data.txt" using 1:3 title 'Column 3'


plot  "data.txt" using 1:2 title 'Column 2'
gnuplot> set term png             (will produce .png output)
gnuplot> set output "printme.png" (output to any filename you use)
gnuplot> replot
```