Profiling
Topics

- How can we find what code takes the most time?
- How can we inspect a compiled executable?
Profiling:
time & gprof
• Use time for how long a program takes to run:

```bash
# time ./myapp ..
real 0m7.546s
user 0m0.006s
sys 0m0.016s
```

! # time ./timing
! # time ./timing busy
Waiting

• Options to slow down a program:
  - Calling kernel sleep functions:
    ```
    ...
    ```
  - Busy waits, like:
    ```
    for (int i = 0; i < 20000000; i++) {
      // Do nothing
    }
    ```

• Busy wait is bad:
  - Consumes CPU time: not given to other threads
  - Consumes power: CPU runs at max speed
  - Time of delay..
  - Non-portable: changes with different CPU / compiler
Profiling with gprof

- What parts take the most time?

**gprof Usage:**
- Enable with GCC flag: `-pg`
- Log written to current directory when program exits
  - Log named gmon.out
- Analyze log with one of:
  
  ```bash
  # gprof myApp gmon.out
  $ arm-linux-gnueabihf-gprof myApp gmon.out
  ```
# ./primer
... program runs and exits gracefully, writing gmon.out ...

# gprof primer gmon.out

<table>
<thead>
<tr>
<th>index</th>
<th>% time</th>
<th>self</th>
<th>children</th>
<th>called</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>54.0</td>
<td>10.87</td>
<td>0.00</td>
<td>__aeabi_uidiv [1]</td>
<td>&lt;spontaneous&gt;</td>
</tr>
<tr>
<td>[2]</td>
<td>14.8</td>
<td>2.98</td>
<td>0.00</td>
<td>__udivdi3 [2]</td>
<td>&lt;spontaneous&gt;</td>
</tr>
<tr>
<td>[3]</td>
<td>10.3</td>
<td>0.00</td>
<td>2.08</td>
<td>findPrimesThread [3]</td>
<td>&lt;spontaneous&gt;</td>
</tr>
<tr>
<td></td>
<td>2.07</td>
<td>0.00</td>
<td>16588/16588</td>
<td>isPrime [4]</td>
<td>&lt;spontaneous&gt;</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
<td>0.00</td>
<td>754/754</td>
<td>storeNewPrime [9]</td>
<td>&lt;spontaneous&gt;</td>
</tr>
<tr>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>754/4406</td>
<td>sleep_usec [25]</td>
<td>&lt;spontaneous&gt;</td>
</tr>
</tbody>
</table>

<...>
Profile with GPIO

- Set bit (pin) when entering region of interest
- Clear bit (pin) when leaving region.

- Use oscilloscope or logic analyzer to view actual pin changes.

- May be most useful within kernel or bare-metal due to sys-call overheads changing timing.
Information from Executables
LDD, readelf
LDD

- LDD:..
  - Helps find needed (missing?) libraries on system.
  - Linux libraries are .so files: shared object

```bash
# ldd ./primer
linux-vdso.so.1 (0xbea79000)
libpthread.so.0 => /lib/arm-linux-gnueabihf/libpthread.so.0 (0xb6f68000)
libm.so.6 => /lib/arm-linux-gnueabihf/libm.so.6 (0xb6ef3000)
libc.so.6 => /lib/arm-linux-gnueabihf/libc.so.6 (0xb6e03000)
/lib/ld-linux-armhf.so.3 (0x7f5be000)
```

- Note the folder of the .so file:
  - `/lib/arm-linux-gnueabi/` Emulated floating point
  - `/lib/arm-linux-gnueabihf/` Hardware floating point
readelf

- Displays information on ELF executable files
  - ELF: Executable and Linkable Format

# readelf -h ./primer

ELF Header:
- Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
- Class: ELF32
- Data: 2's complement, little endian
- Version: 1 (current)
- OS/ABI: UNIX - System V
- ABI Version: 0
- Type: EXEC (Executable file)
- Machine: ARM
- Version: 0x1
- Entry point address: 0x10d89
- Start of program headers: 52 (bytes into file)
- Start of section headers: 42464 (bytes into file)
- Flags: 0x5000400, Version5 EABI, hard-float ABI

...
Summary

- Profiling:
  - time to see how much time is used
  - gprof to see where time is used

- Info on Executables:
  - ldd to see what libraries are loaded
  - readelf to see executable's architecture etc.