Lab 1

Getting Started

SE 4000 011

Dr Schilling

Claudia Poptile & Roberto Garcia

September 13th, 2022

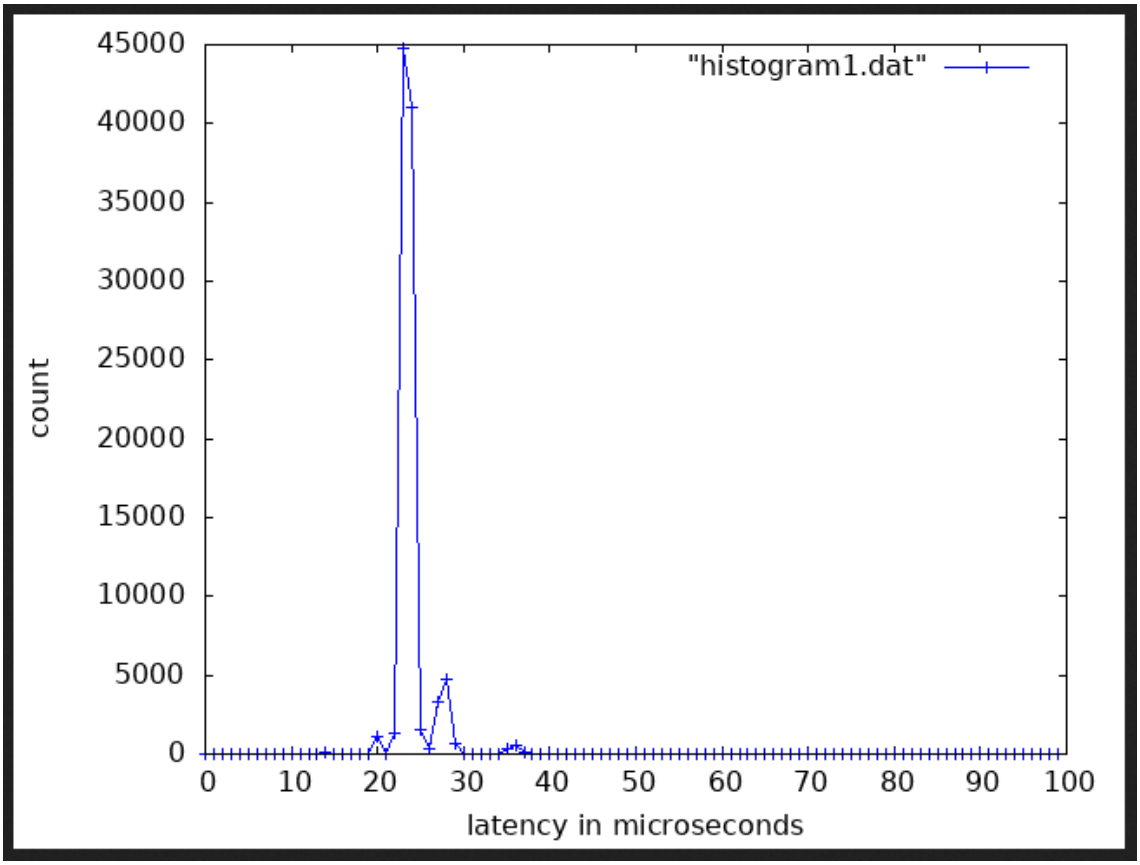
Introduction

This lab served as an introductory lab to get familiar using a Raspberry Pi and a virtual machine. In this lab, 3 experiments were run across the base Raspberry Pi, the Raspberry Pi Real Time Extensions, and the Linux VM to see how deterministic the response time is on each system. The cross compiler on the Virtual Machine was also tested by running simple C programs. A remote debugger was also set up.

Experimental Results

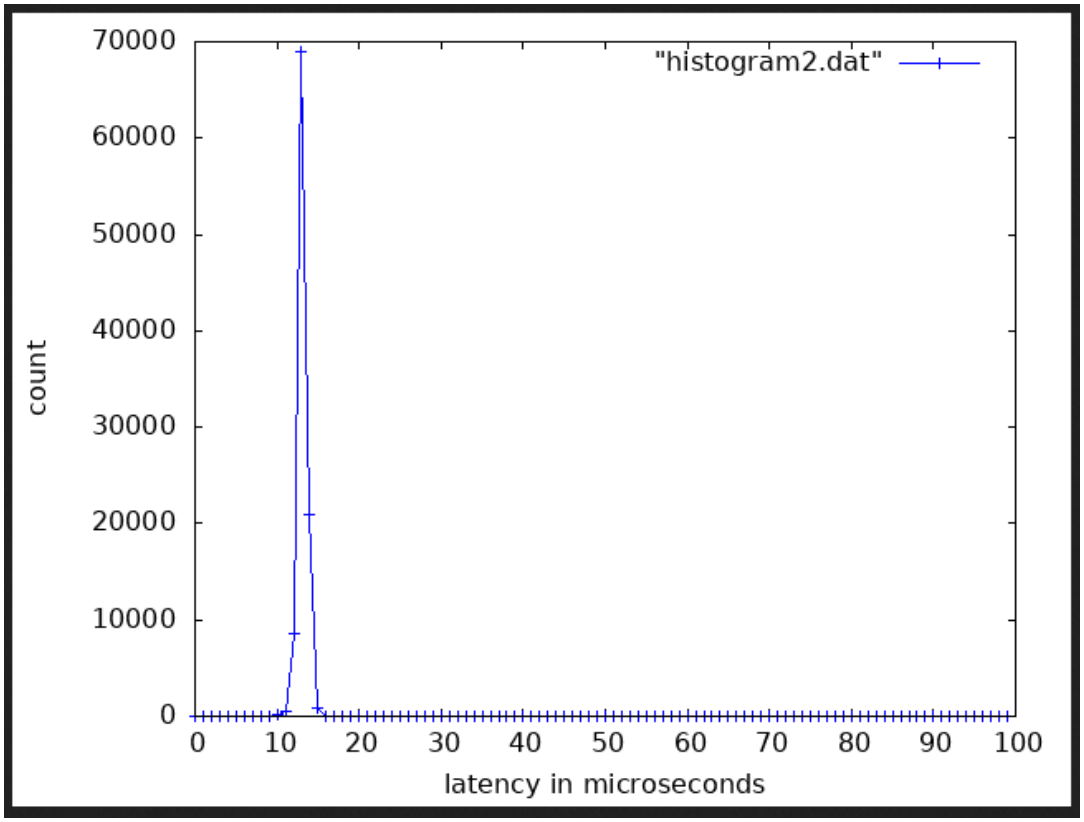
Base Raspberry Pi

```
pi@poptilepi:~ $ sudo uname -a
Linux poptilepi 5.15.61-v7+ #1579 SMP Fri Aug 26 11:10:59 BST 2022 armv7l GNU/L
nux
```



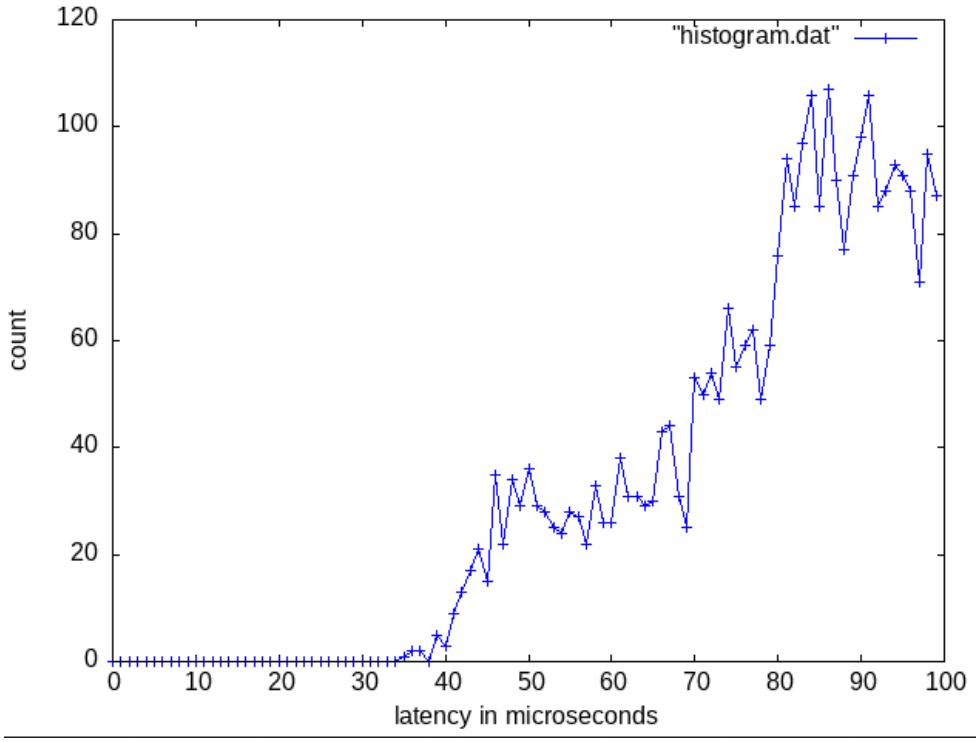
Raspberry Pi Real Time Extensions

```
pi@poptilepi:~ $ uname -a
Linux poptilepi 5.15.40-llat-v7l+ #1 SMP PREEMPT Sat May 21 11:00:21 BST 2022 ar
mv7l GNU/Linux
```



Linux VM

```
pi@poptilepi:~ $ uname -a
Linux poptilepi 5.15.40-llat-v7l+ #1 SMP PREEMPT Sat May 21 11:00:21 BST 2022 ar
mv7l GNU/Linux
```

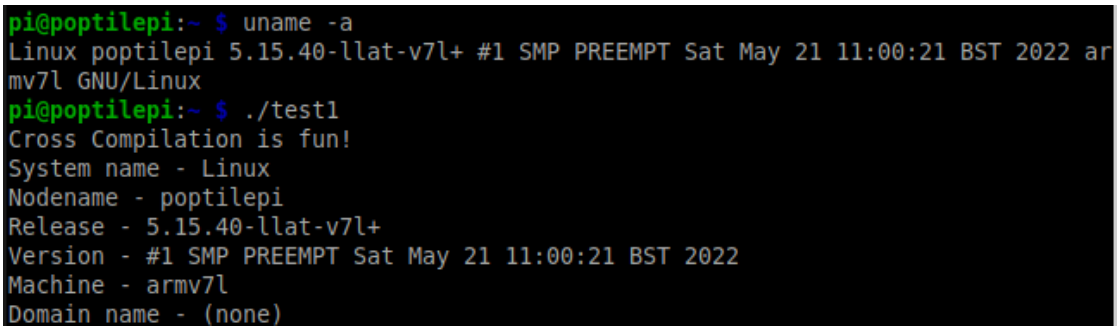


Of the three, which is best suited for real time systems development and operation? Explain why using statistical terms like average, variance, etc.

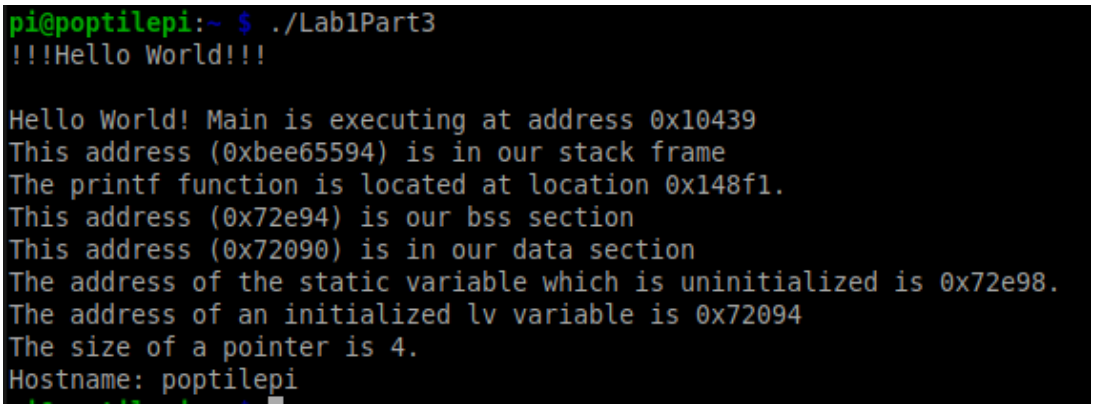
The raspberry pi with real-time extensions would be the preferable environment for real-time systems seeing as the graph shows an average latency to be just above 10 ms while the next closest would be just the base raspberry pi with an average of just over 20ms. Another reason would be the low variance present in the distribution - the pi without the extensions could have unpredictable results anywhere from 20ms to just under 40ms. The Linux VM also has somewhat unpredictable results and a larger distribution overall. Using something with such variability will make estimations and system reliability hard to manage, while the real-time extensions allow for a more predictable and lower overall latency. A lower/tighter distribution, in this case, implies a lower variance and standard deviation for the pi with Real Time Extensions, making the latency somewhat constant and the system more deterministic in terms of response time.

Screen Captures

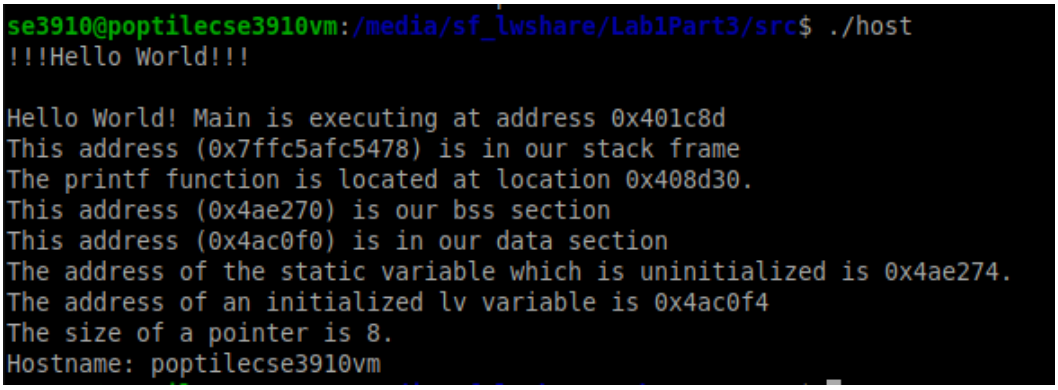
The screenshot below shows the uname command and the execution of the first demo program on the Raspberry Pi.



The screenshot below shows the execution of the second demo program running on the Raspberry Pi.



The screenshot below shows the execution of the second demo program running on the Virtual Machine.



The screenshot below shows the first demo program successfully running on the Raspberry Pi. The original program was modified to have `uname_pointer.__domainname` instead of `uname_pointer.domainname` as it caused a compiler error on the VM.

```
pi@poptilepi:~ $ uname -a
Linux poptilepi 5.15.40-llat-v7l+ #1 SMP PREEMPT Sat May 21 11:00:21 BST 2022 ar
mv7l GNU/Linux
pi@poptilepi:~ $ ./test1
Cross Compilation is fun!
System name - Linux
Nodename - poptilepi
Release - 5.15.40-llat-v7l+
Version - #1 SMP PREEMPT Sat May 21 11:00:21 BST 2022
Machine - armv7l
Domain name - (none)
```

Below are screenshots of the size of the file before and after the program running. The size of the file had increased due to it having been cross compiled to the embedded board.

```
se3910@poptilecse3910vm:/media/sf_lwshare/src$ ls -l
total 16
-rwxrwx--- 1 root vboxsf 1314 Sep 13 13:53 arm1.c
-rwxrwx--- 1 root vboxsf 8336 Sep 13 13:53 test1
```

```
se3910@poptilecse3910vm:/media/sf_lwshare/src$ ls -l
total 412
-rwxrwx--- 1 root vboxsf   1314 Sep 13 13:53 arm1.c
-rwxrwx--- 1 root vboxsf 415504 Sep 13 14:05 test1
```

Questions

Why does the test1 file not run in your vm environment?

The test1 file was written on the Linux virtual machine, but targeted the ARM architecture on the Raspberry Pi. Since the platforms are different, it was not able to run in the VM.

What does the -static option do to the compiler/linker?

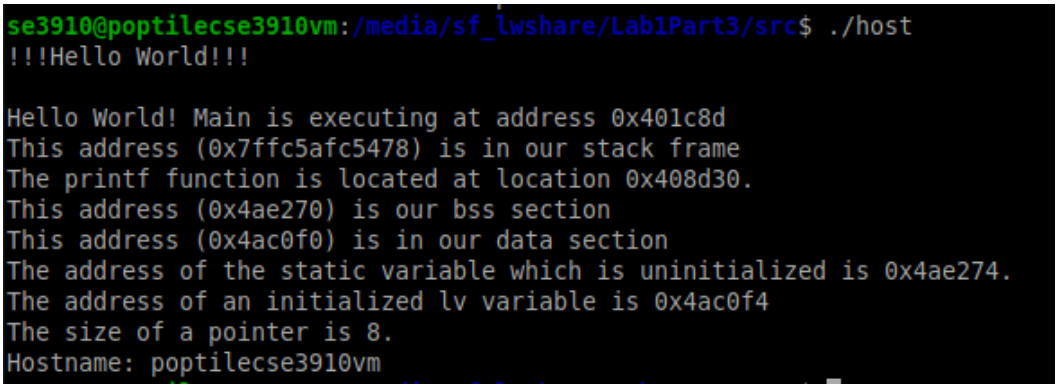
The -static option does not allow sharing of libraries. All linkings are done at compile time and no dynamic linking occurs.

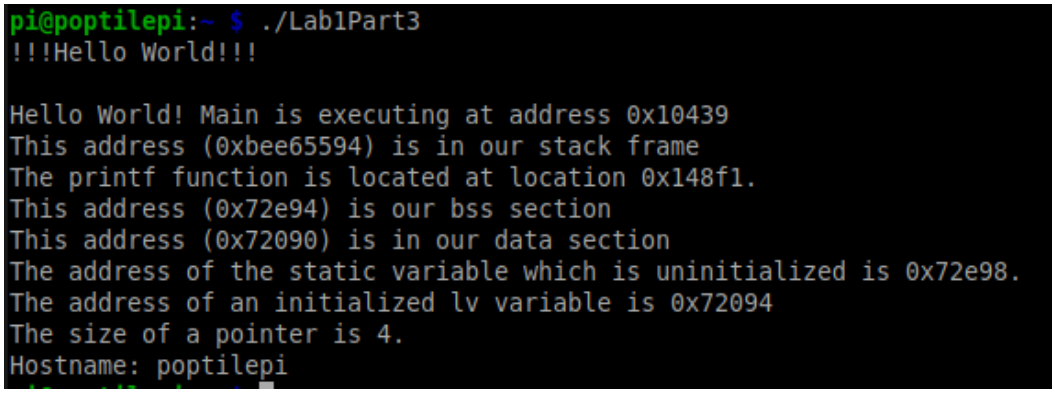
What is the name of the actual binary that performs the cross compilation? What are the name(s) of the links to that binary that help to simplify things? What Linux command might you use to create more of these shortcuts at the OS level?

The binary file is called arm-linus-gnueabihf-gcc-10 while the names of the links are arc-gcc and arm-linux-gnueabihf-gcc. The linux command you might use is “link” or “linkat”

What is different about the addresses given to the variables on the Raspberry Pi versus the host for the second demo? Include screen captures of the address demo program (memory display) running on both the development host and the Raspberry Pi.

The main difference is the address length - the host seems to have a 6 hex character address while the Pi has a 5 character length.





Why is debugging harder on the raspberry pi using gdb than it was when you debugged code with the debugger in your C/C++ Programming course?
In order to debug on the Pi, the VM needs to communicate with the program running on the Raspberry Pi during each step of the debugging process. In our other programming courses we debugged on the same system that the program was running so there were less hoops to jump through.

What is meant by multi-architecture in terms of gdb?
Multi-Architecture is a specific variant of the gbd that allows for debugging on different architectures than the host machine, such as debugging ARM Linux under Intel x86.

What does the gdb service do on the raspberry pi?
It sets up a server on the raspberry pi for the linux host to communicate with for debugging purposes. It allows for remote debugging on the Raspberry Pi by connecting to a program with a remote GDB via target remote.

Conclusions

What have you learned with this experience?

From this experience we learned how to communicate and develop programs using systems with differing architectures. We also generally learned how to use a raspberry pi from burning rasbian images to debugging from a linux development environment. In a more real-time systems sense, we also learned how fast we can expect the pi to be compared to a laptop.

What improvements can be made in this experience in the future?

I'd say it was generally straightforward, the largest roadblock was the occasional mistype or network issues when trying to connect the pi and the laptop to the same network.