9Back : Making Our Plans Great Again

Maxwell Bland Leon Cheung Kilian Pfeiffer

University of California, San Diego

Abstract

The following paper describes a set of benchmarks run on the Plan 9 operating system. In particular, it uses the only actively developed branch, 9FRONT "RUN FROM ZONE!" (2018.09.09.0 Release). The goal of this project is to provide researchers and enthusiasts with greater insight into the current state of the operating system's performance and capabilities of interaction with hardware. Through tests of CPU, Memory, Network, and File System operations, we will gain insight on bottlenecks in the system's performance and the interactions between low-level (hardware) and high-level (OS) system components. These performance statistics will be contrasted with subjective experiences of "responsiveness".

1 Introduction

Plan 9 from Bell labs is a distributed operating system which emerged around the 1980's. It built upon the ideas of UNIX, but adopted an ideology that "everything is a file". Although the system was marketed in the 90's, it did not catch on, as prior operating systems had already gained enough of a foothold. Eventually, during the 2000's, Bell Labs ceased development on Plan 9, meaning official development halted. Unofficial[1] development continues on the 9front fork of the codebase, with new support for Wi-Fi, Audio, and everything anyone could ever want or need.

The experiments were performed as a group using a shared codebase and a single machine described in the following section. The measurements were performed via programs written in Plan 9's *special* version of C 99 and Alef, both described in the original Plan 9 paper cite. The compilers used were the x86 versions of Plan 9's C compiler and Alef compiler, 8c and 8al respectively. The compilers were run with no special optimization settings. Version numbers are not available. Measurements were performed on a single machine running Plan 9 directly from hardware; given the nature of the Plan 9 system, additional metrics could be established for networked file systems and CPU servers; these measurements were not done for sake of simplicity, and because results under these conditions should be inferrable from the results cataloged within this paper.

[1]This is debatable. If you adopt an orphan, are they not your official child?

2 Machine Description

We ran this beautiful operating system of the gods on a Thinkpad T420, the machine of true developers.

```
Processor: model, cycle time, cache sizes (L1, L2,
  instruction, data, etc.)
  Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz
  cache size 3072 KB
  cpu family 6
  model 42
  stepping 7
  siblings 4
  cores 2
  fpu yes
  fpu_exception yes
  fpu_exception yes
  bogomips 4986.98
  clflush size 64
  cache_alignment 64

Memory bus
  DDR3-1333
  i/o-bus-frequency: 666MHz
  bus-bandwith:  10656 MB/s
  memory-clock:  166MHz
  Column Access Strobe (CAS) latency:

I/O bus
  SataIII-speed: 600MB/s

RAM size
  8 GB
```

```
Disk: capacity, RPM, controller cache size
  Samsung SSD 860 EVO 500GB
  Capacity: 500GiB
  RPM: 550MB/s read, 520 MB/s write
   and 98,000 IOPS (Read QD32)
  Controller Cache Size: 512MB
Network card speed:
 Intel 82579 LM Gigabyte: 1Gb/s
 intel Centrino Ultimate-N 6300: 450 Mbps

Operating system (including version/release)
  9FRONT "RUN FROM ZONE!" (2018.09.09.0 Release)
```

3 Experiments

For each section, we report the base hardware performance, make a guess as to how much overhead software will add to base hardware performance, combine these two into an overall prediction of performance, and then implment and perform the measurement. In all cases, we run the experiment multiple times, and calculate the standard deviation across measurements. We use the `nsec()` syscall to record the timestamp. Dynamic CPU frequency scaling was disabled for all trials, and all trials were restricted to a single core.

3.1 Measurement Overhead

The following section reports overheads of reading time, and the overhead of using a loop to measure meany iterations of an operation. One trial involves looping over 16 `nsec` timing calls, 2^{16} times. We average over a single trial, and we do 64 trials.

Present results Discuss Cite the source for the base hardware performance. Compare the measured performance with the predicted performance. If they are wildly different, speculate on reasons why. What may be contributing to the overhead? Evaluate the success of your methodology. How accurate do you think your results are? For graphs, explain any interesting features of the curves. Answer any questions specifically mentioned with the operation. State units of all reported values

3.1.1 Hypothesis

Here's a typical figure reference. The figure is centered at the top of the column. It's scaled. It's explicitly placed. You'll have to tweak the numbers to get what you want.

This text came after the figure, so we'll casually refer to Figure 1 as we go on our merry way.

Figure 1: Wonderful Flowchart

3.2 New Subsection

It can get tricky typesetting Tcl and C code in LaTeX because they share a lot of mystical feelings about certain magic characters. You will have to do a lot of escaping to typeset curly braces and percent signs, for example, like this: "The `%module` directive sets the name of the initialization function. This is optional, but is recommended if building a Tcl 7.5 module. Everything inside the `%{, %}` block is copied directly into the output. allowing the inclusion of header files and additional C code."

Sometimes you want to really call attention to a piece of text. You can center it in the column like this:

`_1008e614_Vector_p`

and people will really notice it.

The noindent at the start of this paragraph makes it clear that it's a continuation of the preceding text, not a new para in its own right.

Now this is an ingenious way to get a forced space. `Real *` and `double *` are equivalent.

Now here is another way to call attention to a line of code, but instead of centering it, we noindent and bold it.

```
size_t :  fread ptr size nobj stream
```

And here we have made an indented para like a definition tag (dt) in HTML. You don't need a surrounding list macro pair.

> `fread` reads from `stream` into the array `ptr` at most `nobj` objects of size `size`. `fread` returns the number of objects read.

This concludes the definitions tag.

3.3 How to Build Your Paper

You have to run `latex` once to prepare your references for munging. Then run `bibtex` to build your bibliography metadata. Then run `latex` twice to ensure all references have been resolved. If your source file is called `usenixTemplate.tex` and your `bibtex` file is called `usenixTemplate.bib`, here's what you do:

```
latex usenixTemplate
bibtex usenixTemplate
latex usenixTemplate
latex usenixTemplate
```

3.4 Last SubSection

Well, it's getting boring isn't it. This is the last subsection before we wrap it up.

4 Acknowledgments

A polite author always includes acknowledgments. Thank everyone, especially those who funded the work.

5 Availability

It's great when this section says that MyWonderfulApp is free software, available via anonymous FTP from

`ftp.site.dom/pub/myname/Wonderful`

Also, it's even greater when you can write that information is also available on the Wonderful homepage at

`http://www.site.dom/~myname/SWIG`

Now we get serious and fill in those references. Remember you will have to run latex twice on the document in order to resolve those cite tags you met earlier. This is where they get resolved. We've preserved some real ones in addition to the template-speak. After the bibliography you are DONE.

Notes

[1] Remember to use endnotes, not footnotes!