HARVEY MUDD COLLEGE

Computer Science Clinic

Final Report for
SpaceX

Process Time Analysis In Spaceflight

March 12, 2015

Team Members
Wendy Brooks (Fall Project Manager)
May Lynn Forssen (Spring Project Manager)
Alix Joe
Rachel Macfarlane

Advisor
Ben Wiedermann

Liaisons
Jesse Keller
Jessica Hester
Jim Gruen

Chapter 1

Background

1.1 Introduction

SpaceX is an American space transport company that designs and manufactures advanced rockets and spacecrafts. The eventual goal of the company is to allow people to live on other planets. SpaceX rockets depend on a variety of software to be successful, including simulations, flight software, and data analysis. Almost of this software is written in-house and is complex enough that subtle problems are often hard to catch and very time consuming to debug.

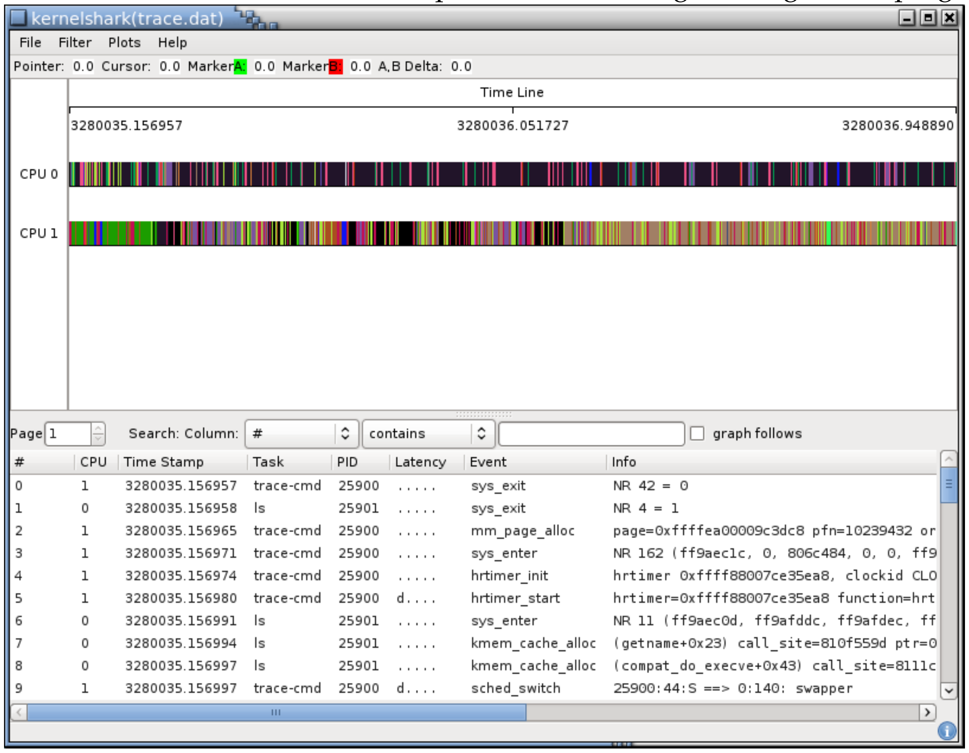
1.2 Tracing and Debugging

The data that SpaceX developers examine in order to find software bugs is called a kernel trace, which documents the different programs running on a computer over a certain span of time, keeping track of when each program was running on each processor.

In order to find potential bugs in their software, SpaceX developers look for problems like unexpected preemptions, priority inversions, and breaks in the pattern of programs being run. A preemption occurs when one program is running, and is then blocked by another program starting to run on the same CPU. A priority inversion occurs when a high-priority program is indirectly blocked by something of lower priority using up a shared resource that the higher priority program needs in order to run. The software running on SpaceX rockets has deadlines for programs at regular intervals because timing is very important in rocket flight, so when there is a break in this cyclical pattern of processes this means that there is likely a bug in the software.

1.3 Existing Tool

SpaceX currently uses a tool called kernelshark to visualize this trace data and help find any bugs or anomalies. We were informed by our liaisons that this debugging process can take up to several days, so searching for bugs using this tool uses up a lot of valuable time. Much of the user expeience of kernelshark is outdated or could use improvements. The kernelshark tool consists of a graphical display of the programs running on a computer over time, as well as a tabular view of the same information, in the form of CPU events such as processes switching, waking, or sleeping.



#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	1	3280035.156957	trace-cmd	25900	.....	sys_exit	NR 42 = 0
1	0	3280035.156958	ls	25901	.....	sys_exit	NR 4 = 1
2	1	3280035.156965	trace-cmd	25900	.....	mm_page_alloc	page=0xffffea00009c3dc8 pfn=10239432 or
3	1	3280035.156971	trace-cmd	25900	.....	sys_enter	NR 162 (ff9aec1c, 0, 806c484, 0, 0, ff9
4	1	3280035.156974	trace-cmd	25900	.....	hrtimer_init	hrtimer 0xffff88007ce35ea8, clockid CLO
5	1	3280035.156980	trace-cmd	25900	d....	hrtimer_start	hrtimer=0xffff88007ce35ea8 function=hrt
6	0	3280035.156991	ls	25901	.....	sys_enter	NR 11 (ff9aec0d, ff9afddc, ff9afdec, ff
7	0	3280035.156994	ls	25901	.....	kmem_cache_alloc	(getname+0x23) call_site=810f559d ptr=0
8	0	3280035.156997	ls	25901	.....	kmem_cache_alloc	(compat_do_execve+0x43) call_site=8111c
9	1	3280035.156997	trace-cmd	25900	d....	sched_switch	25900:44:S ==> 0:140: swapper

Using only this visualization, it can be difficult to see where the cyclical deadlines for the programs begin and end, making it hard to find when something misses a deadline. In addition, the tabular display of the CPU events in the bottom part of the screen has minimal search functionality and the different columns of the table are not sortable.

1.4 Problem Statement

Our team will make it easier and faster for SpaceX engineers to find software problems by creating a tool that visualizes rocket data and highlights problem areas. We will create an improved user experience and provide more visualizations of the data than the old tool.

Chapter 2

SpaceShark

2.1 Design

2.2 Architecture

2.3 Parser

Chapter 3

User Feedback

3.1 General Feedback

3.2 Feedback on Specific Pages

Chapter 4

Challenges

4.1 trace-cmd versioning

As already described, we collect data using a command-line utility called trace-cmd. trace-cmd records various events in the Linux kernel while it runs and then outputs this information into a data file with a particular binary format. trace-cmd also has a "report" option, which takes an existing trace file and generates a human readable report of this record. Using this format as a starting point instead of the raw binary format simplifies parsing. We decided to leverage trace-cmd report instead of wrestling with the binary data ourselves. This allowed us to make progress faster and to avoid working on an already solved problem.

Unfortunately, the format of the file generated by trace-cmd report is dependent on the version of trace-cmd used. For example, the formatting of the "extra info" field of a switch event is liable to change. We use this field to identify preemptions. To be able to reliably parse data and add additional information about preemptions, we need our input data to have a consistent format. This creates a dependency on a particular version of trace-cmd. Our liaisons have been using trace-cmd version 2.2.1, so our parser expects input data to be in the format of 2.2.1.

4.2 Selection of charting tool

We chose to use web technologies because we felt there was a wealth of charting and UI/UX tools that we could use. We spent several weeks experimenting with different charts for the main page. The main page renders a timeline of running tasks, which is essentially a large number of colored bars. This chart needed to be responsive to zooming and dragging, and to dealing with click events to move to a particular time in the table of events displayed below it.

4.3 Batch loading

Chapter 5

Future Work

5.1 Known bugs

5.2 Extensions

5.3 Scalability

5.4 more user testing - results