Trigger Scripts For Extensible File Systems

Cameron Macdonell

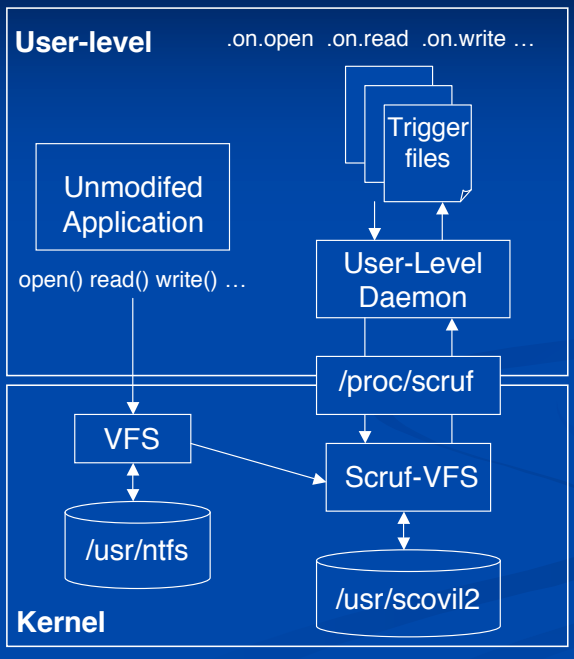
Overview

- The Scruf Framework
- Examples of Extended File Systems
- Experiments
- Concluding Remarks

Scruf - SCRiptable User-level File system

- Scruf executes a user script as a user-level process whenever a file operation occurs in the directory (CWD) containing the script
  - Scruf passes the name of the file being accessed as a parameter (`$1`) to the script
  - Locates a script based on its name - `.on.open` for example
- Allows extension without changing the applications

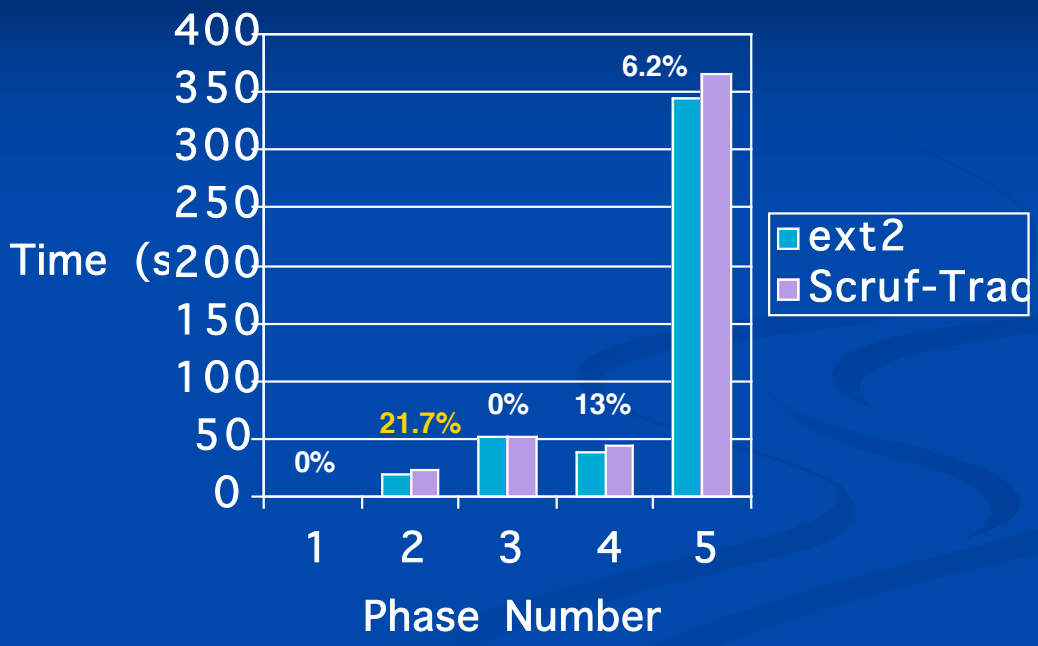
The Scruf Framework: A Diagram



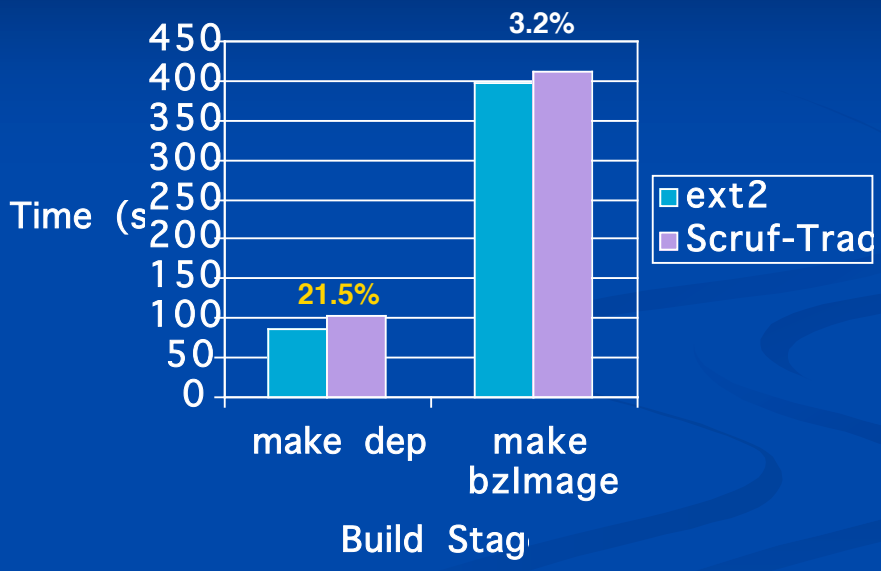
Example Extended File Systems

- We have built five sample extended file systems:
  - Scruf-CVS
    - Backs up files with open-close semantics
  - Scruf-Trace
    - Logs all file operations
  - Scruf-Crypt
    - Transparent access to encrypted files
  - Scruf-Compress
    - Transparent access to compressed files
  - Scruf-Trellis
    - Remote Data Access
- Average 40 lines of code

Andrew100



Build of the Linux Kernel



Running Scruf with No Scripts

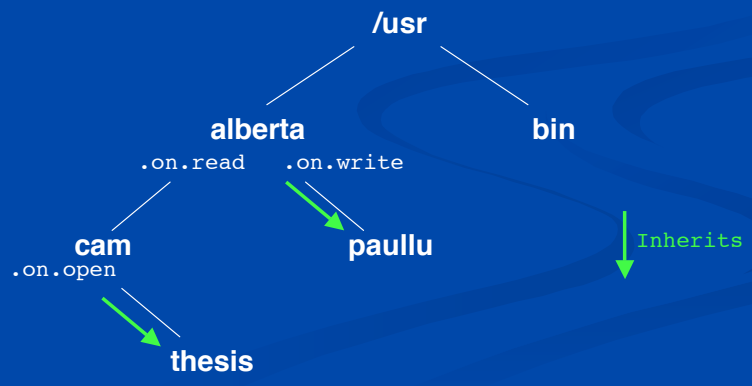
- Measures the overhead of the framework
  - Applications should not be hindered if no extensions are used
- Framework is in place,the daemon is running, there are no scripts in the hierarchy
- What operations are involved?
  - IPC to user-level
  - Search for, and attempted inheritance of, Scripts
  - Filters are setup in kernel
- The highest overhead noted is 0.91%
  - Overheads are a function of the scripts

Concluding Remarks

- Scruf is simple, effective and flexible
  - Useful extensions can be implemented quickly
  - Trigger scripts average 40 lines of code
- Applications do not need to be modified
- Performance
  - Negligible overhead from the framework
  - Asynchronous and Persistent Scripts can improve performance when they can be used

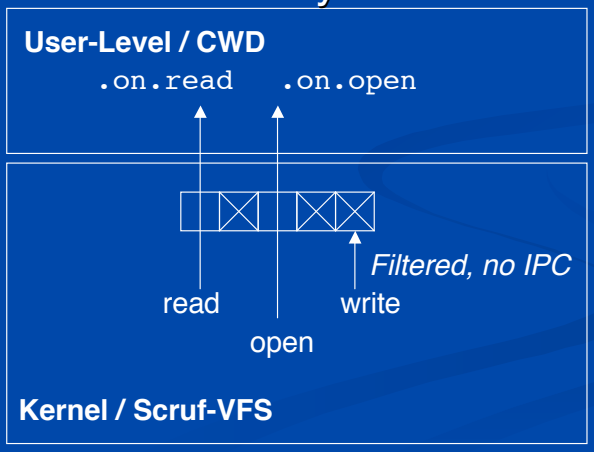
Making Life Easier

- Inheritance
  - Simplifies installation and removal of scripts
  - With inheritance, trigger scripts are inherited into lower directories



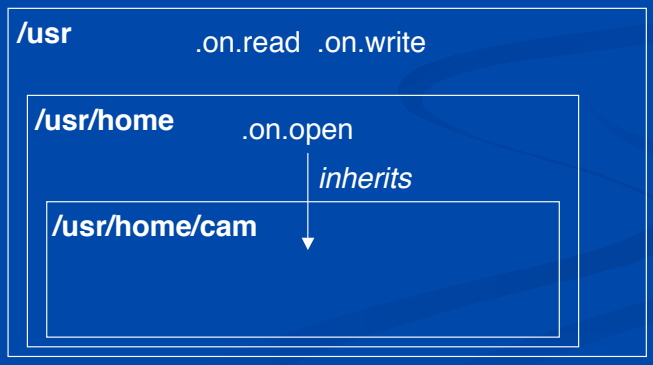
Making Life Easier

- Filtering
  - Keeps track of which directory contain which scripts
  - Eliminates unnecessary IPC



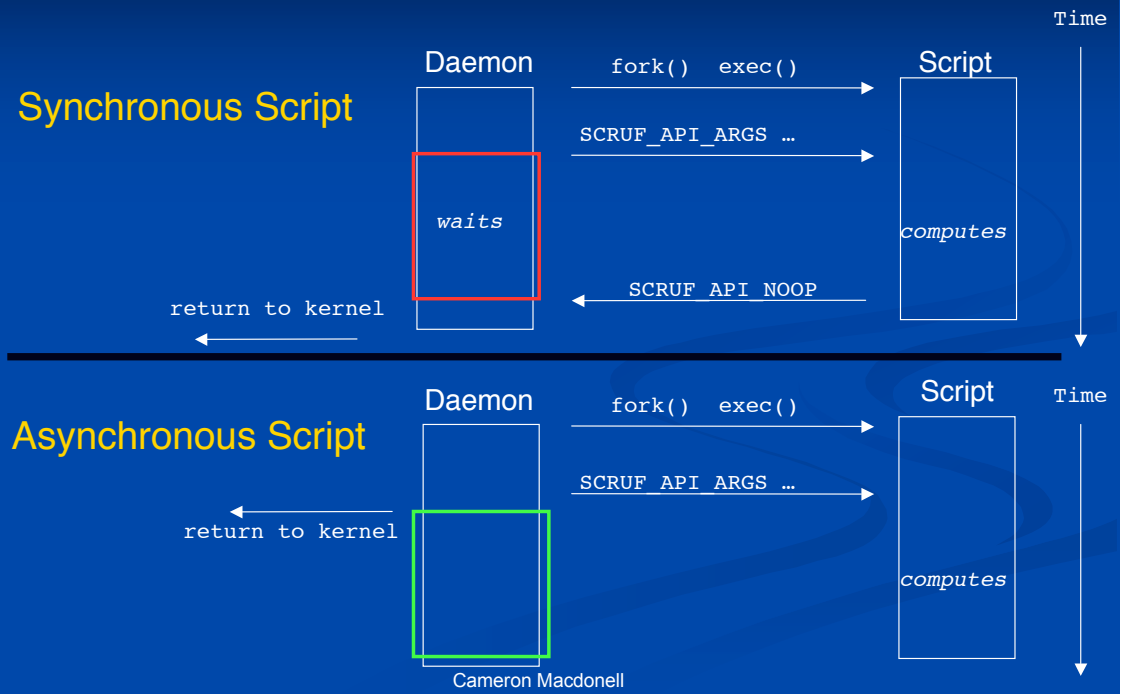
Making Life Easier

- Inheritance
  - Simplifies installation and removal of scripts
  - With inheritance, trigger scripts are inherited into lower directories



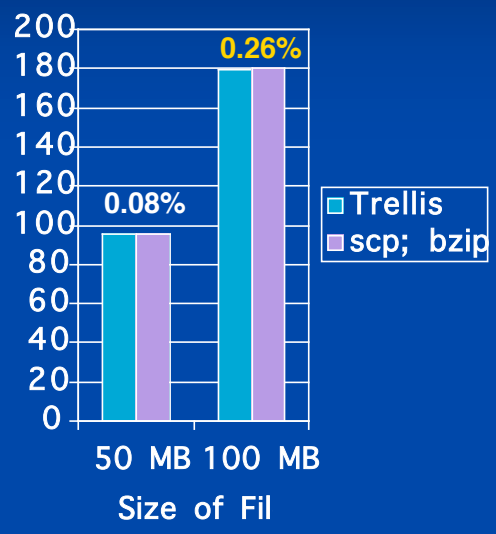
Cameron Macdonell

Sync v. Async Scripts



Cameron Macdonell

Scruf-Trellis



- With development Scruf-Trellis should beat its equivalent
  - Overlapped copying and compression

Making Life Easier

- Symbolic Links

```
[cam@sunset ~/test]$ ls -al
total 16
drwx--S---    2 cam      grad         4096 Aug 15 14:48 .
drwxr-sr-x   69 cam      grad         4096 Aug 15 14:48 ..
lrwxrwxrwx    1 cam      grad            8 Aug 15 14:47 .on.lseek -> .on.open
-rwx------    1 cam      grad           43 Aug 15 14:46 .on.open
lrwxrwxrwx    1 cam      grad            8 Aug 15 14:47 .on.read -> .on.open
lrwxrwxrwx    1 cam      grad            8 Aug 15 14:47 .on.release -> .on.open
lrwxrwxrwx    1 cam      grad            8 Aug 15 14:47 .on.write -> .on.open
-rw-------    1 cam      grad          951 Aug 15 14:48 anyfile
[cam@sunset ~/test]$ cat .on.open
#!/bin/sh

echo $2 >> /usr/scovil3/logfile
[cam@sunset ~/test]$
```

The Scruf Framework: At User-Level

- Run when their corresponding file operation occurs
- Can perform two actions
  1) Interposed Action
     - File operation continues as normal
  2) Redirection Action
     - Redirect Scruf-VFS to a different file
       - Currently, can only come from `.on.open`

Back to Our Example

- For our purpose instead of storing editor preferences, we want to store an executable command
  - `cvs commit` *`filename`*
- Important: we don't want to have to re-write every application to run the script
- Solution: Have the file system look for and execute the scripts when certain file operations (i.e., open) occur

VFS Trigger Scripts

- Allows extension of five file operations
  - `open, read, write, seek, release`
    - release is like close
- Extends these operations by running specially-named, user-defined scripts in the CWD when they occur
  - `open` ⟶ `.on.open`
  - `read` ⟶ `.on.read`
- Passes data describing the operation to the script
  - `/usr/home/cam/myfile.c,read,0,4096,2024,14:24:37.215`

Path Trigger Scripts

- Explicitly invoked in a pathname
  - `trellis:scp:sunset:~/myfile.c`
- Assume path before colon is an executable
- Text following colon is passed on the command line to the path script
- Equivalent to

```
[cam]$ trellis scp:sunset:~/myfile.c
```

The Scruf Framework

- Three main components:
  1) A kernel module, called Scruf-VFS
     - A modified `ext2` module
     - Communicates to user-level when file operations occur
  2) A user-level daemon
     - Searches for and executes the scripts
  3) The trigger scripts
     - Specially named files which encapsulate the functionality which extend the file system's operation
- What does not change?
  - The applications!

The Scruf Framework: At User-Level

- The Daemon
  - A multiplexor in a one-to-many environment
    - *One:* Scruf-VFS in the kernel
      - Communicate with the daemon via /proc file system
    - *Many:* The Trigger Scripts
      - Communicate with the daemon via pipe IPC
  - Locates and runs scripts, if they exist

```
[cam@sunset ~/test]$ ls -al
total 16
drwx--S---    2 cam      grad         4096 Aug 15 14:48 .
drwxr-sr-x   69 cam      grad         4096 Aug 15 14:48 ..
-rwx------    1 cam      grad           43 Aug 15 14:46 .on.open
-rw-------    1 cam      grad          951 Aug 15 14:48 anyfile
[cam@sunset ~/test]$
```

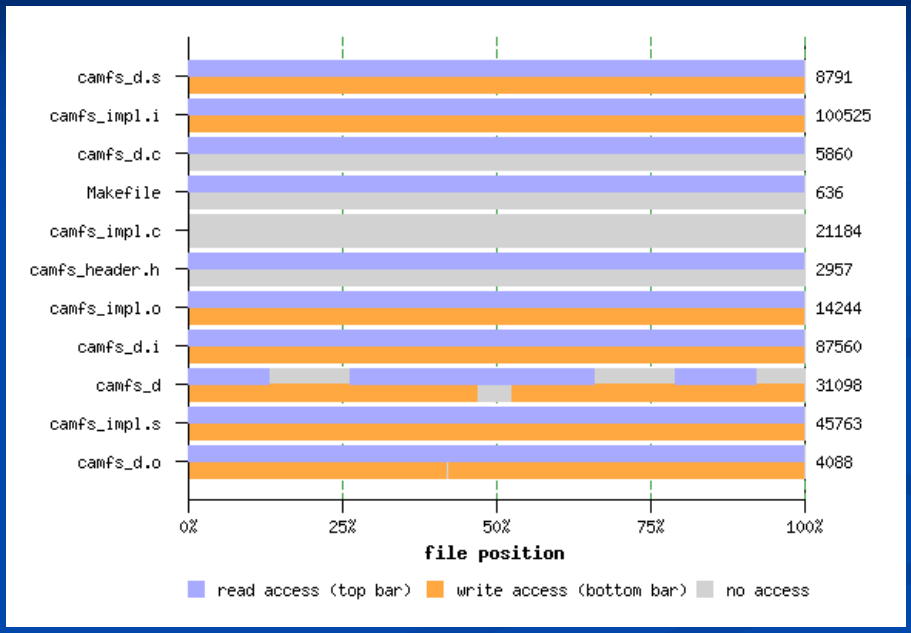
Trigger Scripts

- General Operation
  - Receive data describing file operation
    - Receive data two ways
      - Command line and pipe IPC (`stdin`, `stdout`)
      - `usr/alberta/myfile.c,read,0,4096,1665,16:41:30.716`
  - Perform computation
    - Compression, encryption, write to log file, …
  - Return an code to the daemon which is the passed to the kernel

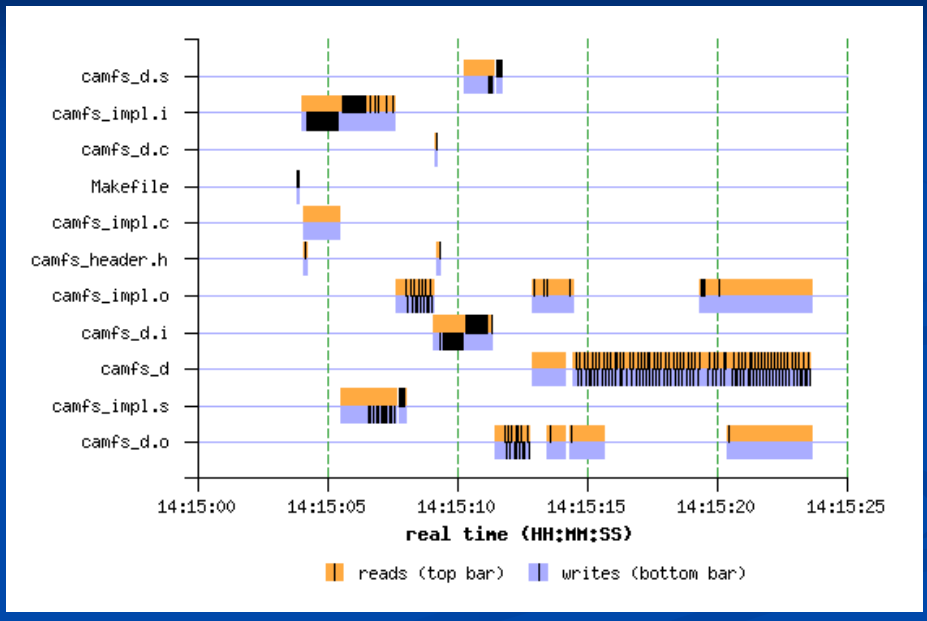
Scruf-Trace

- Helps understand file I/O patterns in applications
- Used to trace the file I/O in `gcc`
  - Traces all opens, reads, writes, lseeks and releases
- Can log thousands of file ops per second
- Each trigger file is 64 lines of compiled C code

Scruf-Trace



Scruf-Trace



The Scruf Framework: In the kernel

- Underneath is exactly like `ext2`
  - Storage and access of files does not change
  - When no trigger scripts are present, Scruf behaves just like `ext2`
- An `ext2` partition can be mounted as a Scruf partition without reformatting
- Changes are restricted to a Linux kernel loadable module
  - No patching, recompilation or rebooting

Making Life Easier

- Four mechanisms which make trigger scripts easier to user and more efficient
  - Symbolic links
  - Persistence
  - Inheritance
  - Filtering

Motivating Example

- File Back-up
  - For software development, paper writing
    - Revisions are created using versioning software like CVS
    - They are recovered in case of lost data, or to “start fresh”
  - Encapsulate the command into a script

```
#!/bin/sh
cvs commit  $1
```

  - Then we need the system to run the script every time a file is opened (or closed).

What are Trigger Scripts?

- Trigger scripts are executable scripts which extend file system functionality
- They are regular user-level programs which can be implemented in any language
  - C, Python, Perl, etc.
- Supported by the Scruf framework
- But, why would anyone want to extend the file system?

Similar Idea: RC files

- A similar idea already exists with RC files
- Editors such as Vim and Emacs have thousands of options
  - i.e., colors, spacing, syntax highlighting
- No user would tolerate inputting these options every time they open a file in the editor
  - Instead, the save their preferences in a file, with a predefined name (.vimrc, .emacs) that the application looks for and reads when it is started.

Making Life Easier

- Persistent Scripts
  - Scripts that do not exit after handling a single file operation
  - Remains running for subsequent operations
    - Saves startup overhead
    - Scruf-Trace logs 2,300 access/second
  - Daemon keeps a pipe open between itself and each persistent script

Experiments

- Scruf-Trace was run on two benchmarks
  - Andrew100
    - Based on the well-known Modified Andrew Benchmark(MAB)
      - Five stages
        - `mkdir, cp, ls -l, wc & grep, gcc`
    - MAB is too small on modern system
      - Andrew100 performs MAB 100 times
  - Build of the Linux Kernel
    - A full compilation of the 2.4.18 kernel

Testing Path Scripts

- Path scripts are a second type of trigger script
  - Explicitly named and executed
- Scruf-Trellis
  - Remote data access
  - Compare

```
[cam]$ bzip2 -c trellis:scp:sunset:/usr/cam/mymail
                                                > /dev/null

[cam]$ scp sunset:/usr/cam/mymail tmp ; bzip2 -c tmp
                                                > /dev/null
```

- Tested on 50 MB and 100 MB files
- Written in 30 lines of Python code

Trigger Script API

- Scripts interact with the Daemon via API
  - Language-independent, using ASCII strings
  - Examples:
    - Behaviour Codes
      - `SCRUF_API_ASYNC, SCRUF_API_PERSIST`
    - Return Codes
      - `SCRUF_API_NOOP, SCRUF_API_RELOAD`
    - Control Codes
      - `SCRUF_API_ARGS, SCRUF_API_KILL`

Making Life Easier

- Asynchronous Scripts
  - Non-blocking scripts
  - Daemon does not wait for a return code from the script
    - Assumes *interposed action* and returns `SCRUF_API_NOOP` to kernel
  - Daemon is not blocked waiting for a return values, instead returns immediately to kernel and retrieves next operation
  - Most useful with persistent scripts