Prospect: A New Performance Tool for Linux

Internals, Operation, Usage

Alex Tsariounov

Hewlett-Packard Company
Linux Systems Division R&D
Fort Collins, Colorado

alex_tsariounov@hp.com

HP WORLD 2002
Conference & Expo

Talk Overview

- Historical Perspective: Where Prospect Came From
- Overall Picture: Architecture of Prospect
- Design Elements: 3 Modules in Detail
- Usage: Simple and Advanced Cases
- Differences to HP-UX Prospect
- Simple Output Walkthrough

What is Prospect?

- Prospect is an instruction-pointer-sampling flat profiler for obtaining code profiles in a non-intrusive way
- One can obtain both symbol-level and assembly-level profiles without undue requirements on the target application
- In other words, there is no need to rebuild or relink the application, and the only requirement is that the applications not be stripped

Historical Perspective

- In the beginning (1988)
  - Prospect was created as a test tool for HPUX Kernel Instrumentation package
    - KI is a kernel tracing facility that still exists today in modern HPUX systems
  - Heavily dependent on the KI for data generation and complex
  - Became a useful and popular tool among the internal HP HPUX community

Historical Perspective (2)

- In the middle (2001)
  - Investigating Linux tracing facilities
    - Roll your own
    - Linux Trace Toolkit (LTT) was promising
      - Small additions would have enabled prospect
  - All impose a patch of the kernel
    - That was not desirable since we wanted the least modification to a system to enable profiling analysis

Historical Perspective (3)

- Now (2002)
  - Module-based solution was ideal
    - Found oprofile in 2001
    - Almost ready at the time (0.0.2 release)
    - Did not require a kernel patch
    - Fit all requirements including NMI sampling
  - Currently prospect supports 0.0.9 through 0.2 versions of oprofile
    - Prospect uses the module only, not the daemon nor oprofile user-space tools

Prospect Requirements

- Low intrusion
    - Minimal run-time overhead during sampling
    - No special build or link requirements on applications
    - No kernel patching required
- Simple to use
    - Simple command will get you profiles
    - No knowledge of oprofile details or performance counters needed
    - Advanced options can be learned as needed
- Interstitial
    - View an interval of time on the system
    - Analysis of applications in familiar /bin/time fashion

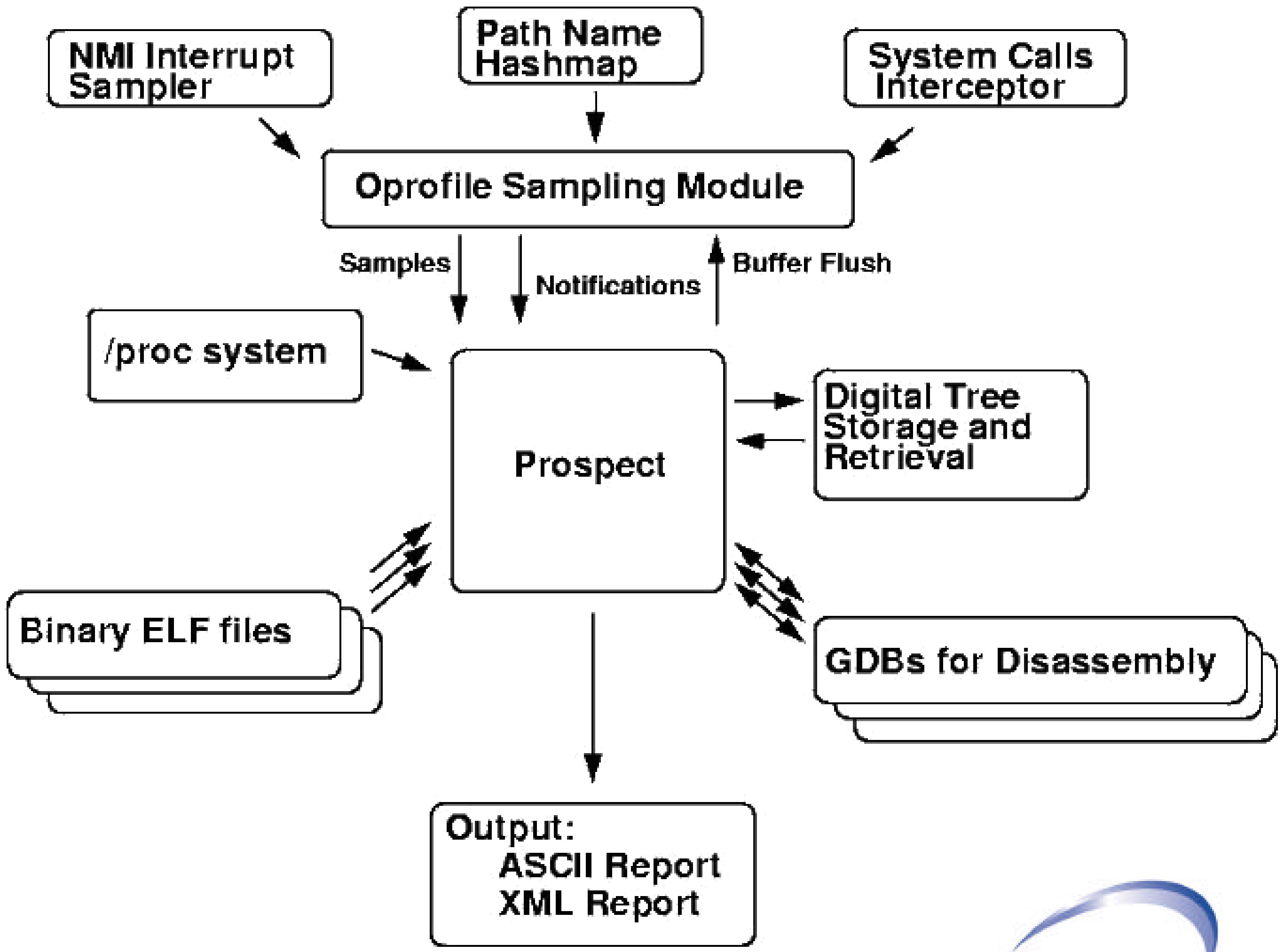
Architectural Elements

- Oprofile Sampling Module
  - Provides three important pieces:
    - NMI-based PC sampling
    - Certain system call interception and trace
    - mmapped file path hashmap
- Prospect System Model
  - All processes on the system are modeled as information structures with a Virtual Address Space of mmapped regions
  - Symbol information is read in after the run
  - The /proc filesystem is used to harvest information about processes already running

Architectural Elements (2)

- Data Storage and Retrieval
  - Management of sparse PC data though use of Digital Tree
  - Also called "Trie" from the word "Retrieval"
  - Part of the data is also the index
- Symbol Mapping with libelf
  - Cross-platform way to get at symbol information in binary files
  - Stripped shared libraries get bracketed symbols (bsearch->qsort)
- GDB for Disassembled Profiles
  - Cross-platform disassembly "server"
  - Configurable effective load

Architectural Summary



Sequence of Execution

- Load and initialize the oprofile sampling module
- Search through /proc and set up structures for all running processes
- Attach to and activate the oprofile module
- Set up periodic alarm signal to flush oprofile buffer
- Exec the child process

Sequence of Execution (2)

- Enter blocking read loop on oprofile device files
- If child has exited, stop profiling and empty oprofile buffer
- Go through the structures of all processes and extract PC hits
- Create the report
- Shutdown, but leave oprofile module loaded

Design Elements: Oprofile Interface

- The oprofile loadable kernel module provides the following capabilities
  - NMI-based sampling on P6 and Athlon processors
  - System call interception and trace of the following system calls
    - fork (and vfork, clone), exec, mmap, init_module, exit
  - mmapped file path hash table for retrieving path information of all mmapped files included in this trace
  - Code is in linux_module.c and linux_module.h File rec_proc.c contains the sequencing for data collection

Elements: Oprofile Interface (2)

- The interface to the module is in the form of /proc control inputs and device file outputs
  - /proc controls buffer sizes, sampling event and frequency setup, buffer flush command, and other information
  - Three device files (in /var) are used in a blocking-read loop to transfer data from kernel to user space
    - opdev: samples file
    - opnotedev: system call trace file
    - ophashmapdev: path hash table file to be memory mapped

Elements: Oprofile Interface (3)

- Phases of use
  - Initialization
    - Prospect first finds the module and loads it if not loaded. Then we open the hashmap device, if successful, we own the device. We then set up a default configuration using CPU_CLK_UNHALTED event with a default frequency of 200 Hz. If the counters were active in the module, prospect does not change them.
    - Prospect next harvests information in /proc for all other processes running on the system and builds the system model.

Elements: Oprofile Interface (4)

– Periodic flush
  - During the run, an alarm signal is used to periodically flush the oprofile kernel buffer. This allows the read that prospect is blocking on to succeed so data is transferred to user space. Frequency is controlled with prospect's -M command line parameter.

– Shutdown
  - Every time a read succeeds, prospect checks for the exit of the child with a waitpid. If true, the oprofile buffer is emptied and the device files closed. (This stops sampling.) The module is left in memory in anticipation of further use.

Design Elements: The Dtree Digital Tree Module

- The PC samples constitute sparse data and prospect uses a Digital Tree to manage storage and retrieval of the data (code is in prospect/dtree)
  - Also called "trie" from the word "Retrieval"
  - Part of the data is also the index
  - A quaternary trie is used which means
    - there can be four items in each node
    - the trie is 16 levels deep for a 32-bit entity
    - five consecutive values will extend the trie to the maximum level

Elements: Dtree (2)

- Traversing the trie to find/insert a value is then done in the following fashion
  - Take top two bits as index into a four element array
  - If the data is present, return it
  - If not, traverse where pointed to then shift by two bits and repeat

  - If a node (the four element array) contains data it is called a "flower"
    - Because of the quaternary nature, a flower can house up to four data elements

Elements: Dtree (3)

- What problem are we trying to solve?
  - The data we store is number of hits indexed by virtual address (the program counter value)
  - We need sorted output at the report stage
  - We have a 32 bit virtual address space
    - This (potentially) has up to 4 billion possible indexes
  - Applicable to every process running on system and kernel

Elements: Dtree (4)

- The interface to this module for inserting and querying is defined in the dtree.h header file as
  - DTI(*Pdt, Idx)
    - Insert data at index Idx in trie pointed to by Pdt
    - Will create entry if first insert
  - DTG(*Pdt, Idx)
    - Lookup data at index Idx in trie pointed to by Pdt
    - Will return NULL if index does not exist (no data there)

Elements: Dtree (5)

- And the interface for extracting all data is defined in the dtree.h header file as
  - DTF(*Pdt, &Idx)
    - Return first element and its index Idx in trie pointed to by Pdt
    - Inclusive of Idx passed in
  - DTN(*Pdt, &Idx)
    - Return next element and its index Idx in trie pointed to by Pdt
    - Exclusive of Idx passed in

Design Elements: Disassembly

- GDB is used as a cross-platform disassembly "server", of sorts, in the post-sampling processing and report generation phase
- Prospect opens a configurable number of pipes to slave GDB processes tied to particular binary files
  - Each GDB process is responsible for one binary file
  - Prospect passes addresses to disassemble and GDB returns disassembled code as a table of strings

Elements: Disassembly (2)

- This is accomplished by "librarizing" GDB in code found in prospect/dass_gdb which has the following interface
  - **dass_open():**
    - fork/exec a GDB process on a particular file
  - **dass():**
    - pass in addresses and returns table of strings of disassembled code
  - **dass_free():**
    - free the block of memory that dass() created
  - **dass_close():**
    - close the file and end GDB process

Elements: Disassembly (3)

- A number of such opened GDB processes (eight by default) are kept open and in a "rolling queue"
  - Once a new file is to be disassembled, prospect first searches the queue for an already opened pipe to a GDB process for that file
  - If found, prospect moves this pipe to the head of the queue
  - If not found, a new pipe is opened to a new GDB process and added to the head of the queue
  - If there are more GDB processes open than queue elements, the one that "falls off" the tail is closed

Elements: Disassembly (4)

- Thus, this code has the following properties
  - The most often used files for disassembly gravitate toward the front of the queue and are found quicker that less often used files
  - The length of the queue determined how many simultaneous pipes to GDB processes are held open and is user-configurable with "-g <number>"
  - Setting -g to 1 (one) will force using only one slave GDB process and save memory but performance will suffer
  - Setting -g to many will improve performance, however memory consumption will go up

Simple Use: To Get a Profile

- Profile immediate child
  - `prospect my_app > output.file`
- Profile immediate and all descendants
  - `prospect -f output.file -V3 my_app`
- Profile everything on the system
  - `prospect -V4 -f output.file sleep 60`
- Errors and diagnostics are output to stderr

  -
    ```
    prospect -V3 -f prospect.output \
    your_app > app.output  2>errors
    ```

Simple Use: Sample Frequency

- Avoiding aliasing
  - Aliasing is a common problem for sampling systems
  - The command line switch "-H <frequency in Hz>" will change sampling frequency for the run
  - For first analysis, run at different frequencies to figure out aliasing
  - Center in on a good frequency from initial runs for further analysis

Advanced Use: Diassembled Profiles

- Command line switch: "-e" will create disassembled profiles
  - Two -e's will force disassembly for all symbols
- Disassembled profiles are useful for determining the "why" of hot functions
- Need to understand the CPU micro architecture in order to account the hits correctly (pipeline effects, etc.)
  - Can then estimate cache and TLB misses

Advanced Use: Kernel Profiles

- Command line switch: "-V k" (exclusive of -V [2-4]) will create kernel profiles in the output
  - NMI-based sampling is imperative for correct kernel profiles
- Four profiles created:
  - Global kernel profile: all kernel hits for everybody
  - Kernel profile due to kernel threads
  - Kernel profile due to user processes
  - Kernel profile of hits to process zero (interrupts)
- Note: The first profile above is the sum of the others

Differences to HPUX Prospect

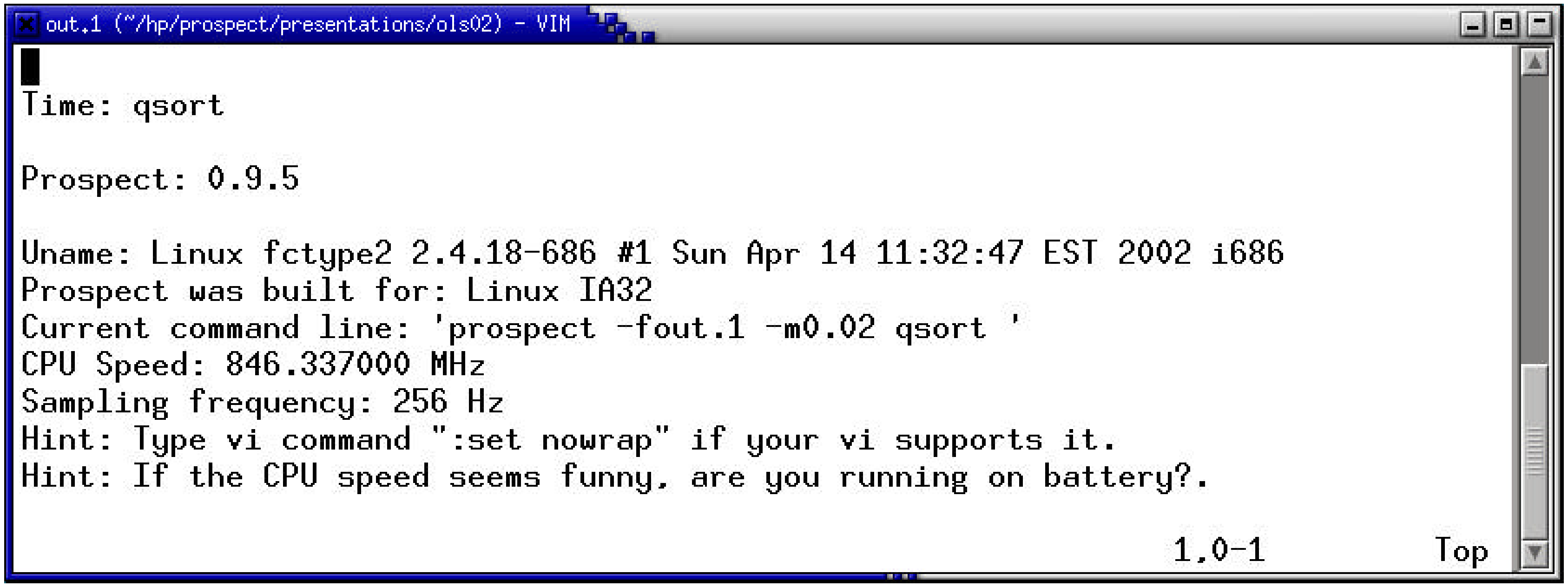
- Things that are worse
  - No system call information
    - Suggest: strace
  - No memory map information
    - Suggest: cat /proc/pid/maps
  - Incomplete features (save to file, etc.)
    - Will fill in with time and demand

Differences to HPUX (2)

- Things that are better
  - Adjustable sampling frequency!
    - Command-line parameter “-H”, 1 Hz to ?....
    - 4000 Hz max tested and approved
  - No missed buffers (at the expense of missed samples)
    - Easier to keep up with trace. Miss singles on overflow
  - Symbols for stripped shared libraries
    - Since statics are gone, we “guess”
    - Qsort hits: **`bsearch->qsort`**

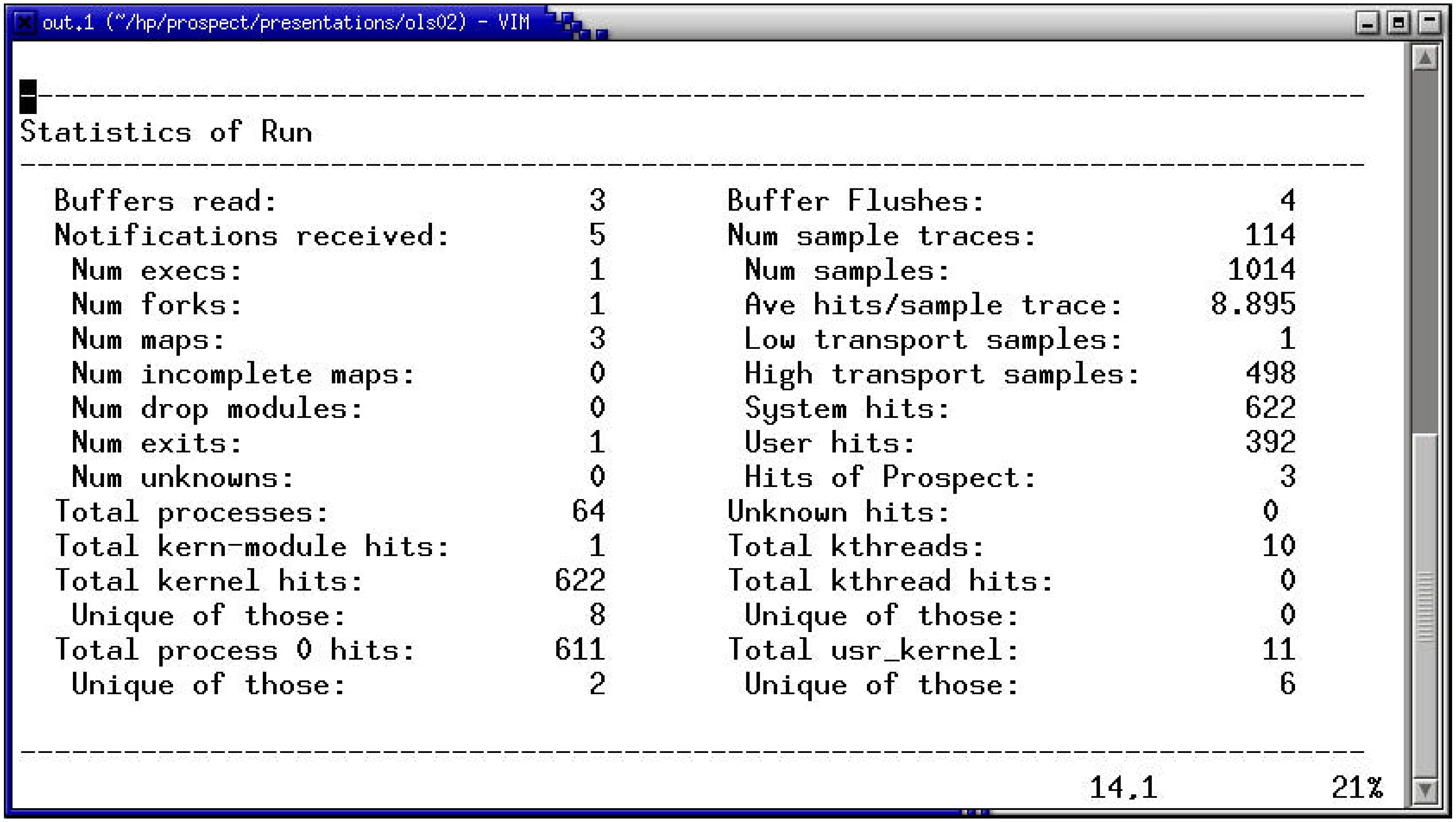
Prospect Output: Simple qsort

- Header information



```
Time: qsort

Prospect: 0.9.5

Uname: Linux fctype2 2.4.18-686 #1 Sun Apr 14 11:32:47 EST 2002 i686
Prospect was built for: Linux IA32
Current command line: 'prospect -fout.1 -m0.02 qsort '
CPU Speed: 846.337000 MHz
Sampling frequency: 256 Hz
Hint: Type vi command ":set nowrap" if your vi supports it.
Hint: If the CPU speed seems funny, are you running on battery?.

                                                         1,0-1         Top
```

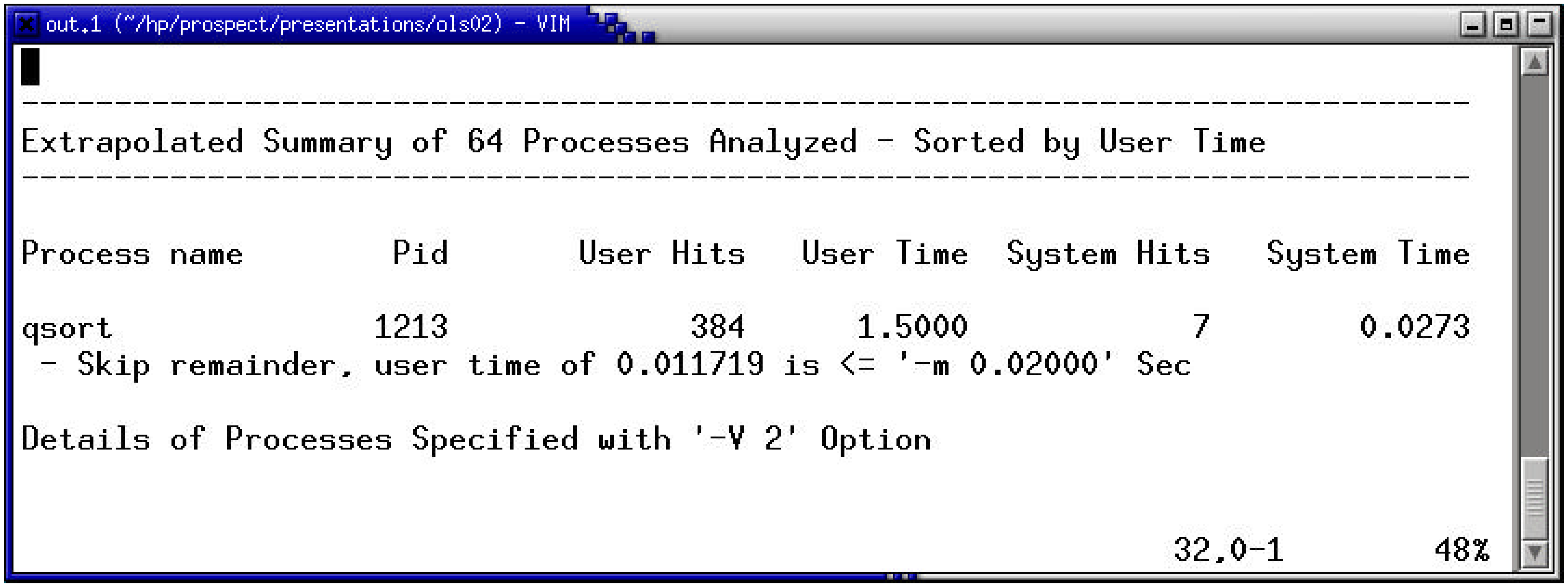
Output: Simple qsort (2)



```
------------------------------------------------------------------------------
Statistics of Run
------------------------------------------------------------------------------
  Buffers read:                  3       Buffer Flushes:                4
  Notifications received:        5       Num sample traces:           114
   Num execs:                    1        Num samples:               1014
   Num forks:                    1        Ave hits/sample trace:    8.895
   Num maps:                     3        Low transport samples:        1
   Num incomplete maps:          0        High transport samples:     498
   Num drop modules:             0        System hits:                622
   Num exits:                    1        User hits:                  392
   Num unknowns:                 0        Hits of Prospect:             3
  Total processes:              64       Unknown hits:                 0
  Total kern-module hits:        1       Total kthreads:               10
  Total kernel hits:           622       Total kthread hits:            0
   Unique of those:              8        Unique of those:              0
  Total process 0 hits:        611       Total usr_kernel:             11
   Unique of those:              2        Unique of those:              6

------------------------------------------------------------------------------
                                                              14,1          21%
```

Output: Simple qsort (3)

- Process table

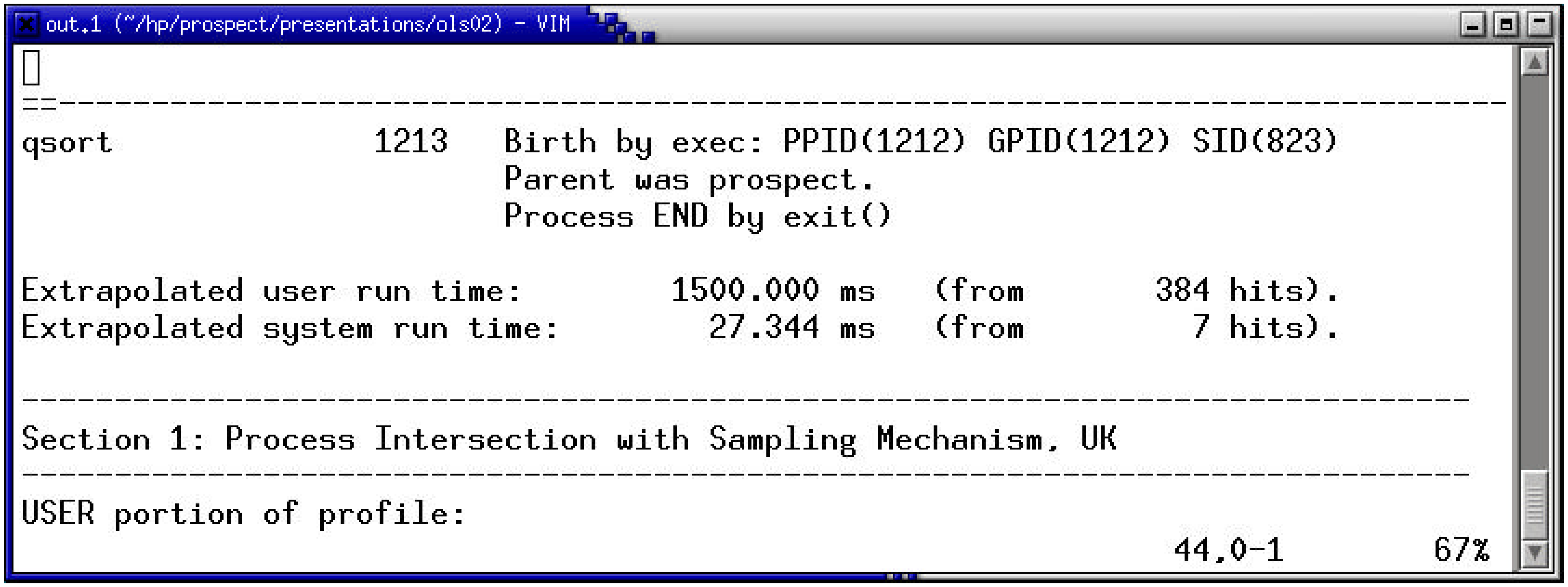


out.1 (~/hp/prospect/presentations/ols02) - VIM

```
--------------------------------------------------------------------------------
Extrapolated Summary of 64 Processes Analyzed - Sorted by User Time
--------------------------------------------------------------------------------

Process name          Pid        User Hits   User Time  System Hits   System Time

qsort                1213              384      1.5000            7        0.0273
 - Skip remainder, user time of 0.011719 is <= '-m 0.02000' Sec

Details of Processes Specified with '-V 2' Option


                                                              32,0-1         48%
```

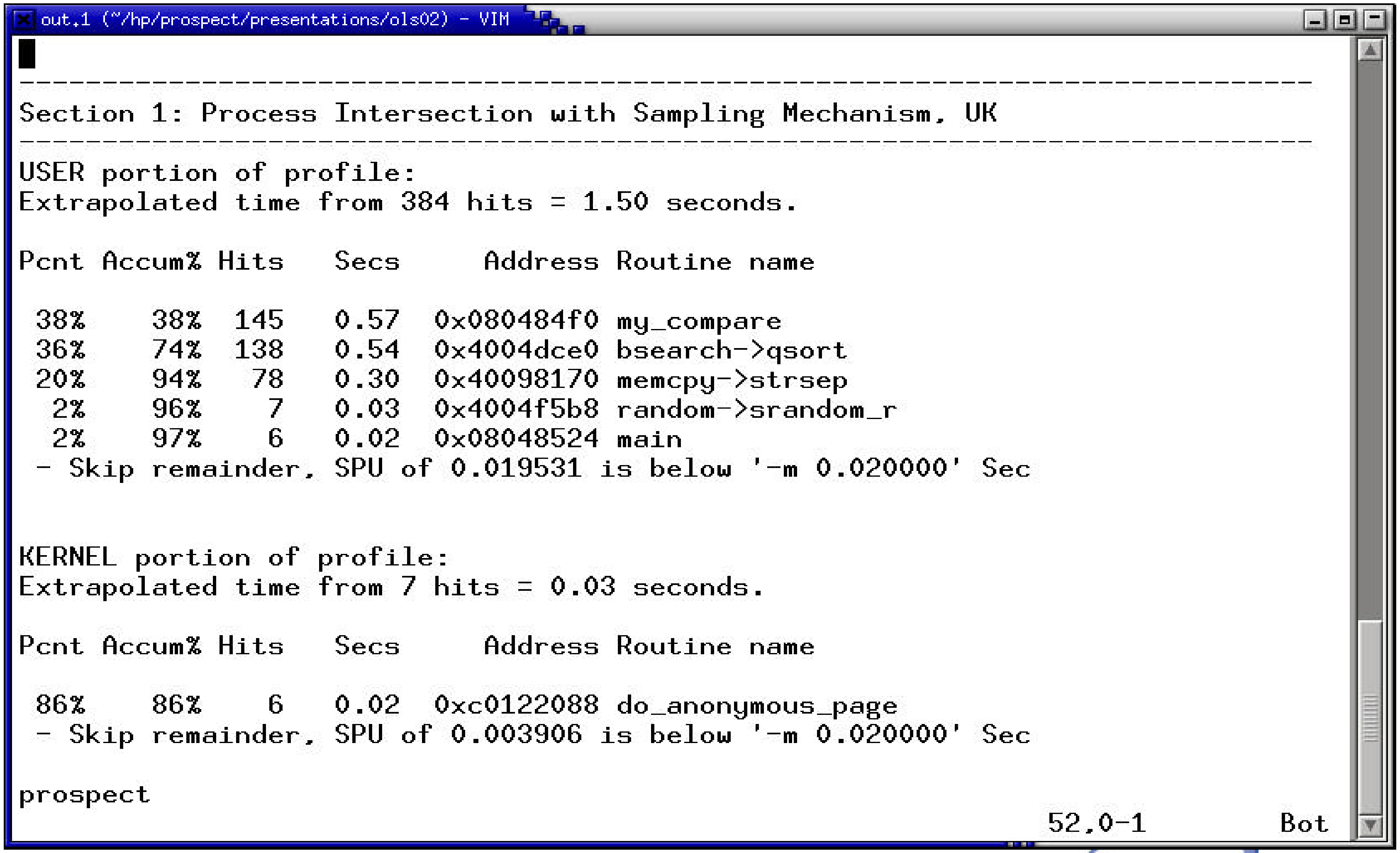
Output: Simple qsort (4)

- qsort process header


out.1 (~/hp/prospect/presentations/ols02) - VIM

```
==---------------------------------------------------------------------------------
qsort                   1213    Birth by exec: PPID(1212) GPID(1212) SID(823)
                                Parent was prospect.
                                Process END by exit()

Extrapolated user run time:        1500.000 ms   (from        384 hits).
Extrapolated system run time:        27.344 ms   (from          7 hits).

-------------------------------------------------------------------------------
Section 1: Process Intersection with Sampling Mechanism, UK
-------------------------------------------------------------------------------
USER portion of profile:
                                                                44,0-1        67%
```

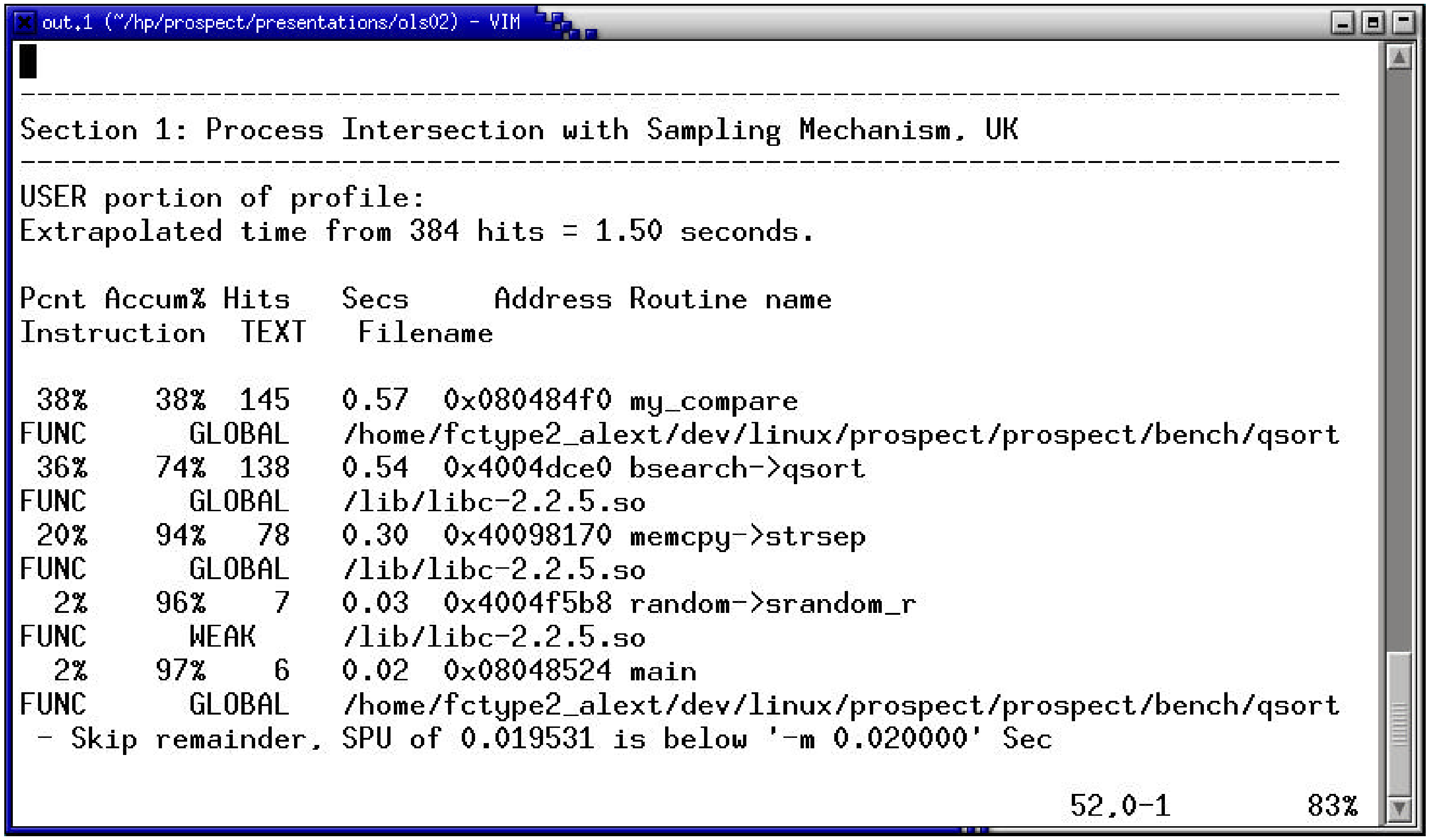
Output: Simple qsort (5)



out.1 (~/hp/prospect/presentations/ols02) - VIM

```
------------------------------------------------------------------------------
Section 1: Process Intersection with Sampling Mechanism, UK
------------------------------------------------------------------------------
USER portion of profile:
Extrapolated time from 384 hits = 1.50 seconds.

Pcnt Accum% Hits   Secs     Address Routine name

 38%    38%  145   0.57  0x080484f0 my_compare
 36%    74%  138   0.54  0x4004dce0 bsearch->qsort
 20%    94%   78   0.30  0x40098170 memcpy->strsep
  2%    96%    7   0.03  0x4004f5b8 random->srandom_r
  2%    97%    6   0.02  0x08048524 main
 - Skip remainder, SPU of 0.019531 is below '-m 0.020000' Sec


KERNEL portion of profile:
Extrapolated time from 7 hits = 0.03 seconds.

Pcnt Accum% Hits   Secs     Address Routine name

 86%    86%    6   0.02  0xc0122088 do_anonymous_page
 - Skip remainder, SPU of 0.003906 is below '-m 0.020000' Sec

prospect
                                                                52,0-1        Bot
```

Output: Simple qsort (6)



out.1 (~/hp/prospect/presentations/ols02) - VIM

```
--------------------------------------------------------------------------------
Section 1: Process Intersection with Sampling Mechanism, UK
--------------------------------------------------------------------------------
USER portion of profile:
Extrapolated time from 384 hits = 1.50 seconds.

Pcnt Accum% Hits   Secs     Address Routine name
Instruction  TEXT   Filename

 38%    38%  145   0.57  0x080484f0 my_compare
FUNC      GLOBAL   /home/fctype2_alext/dev/linux/prospect/prospect/bench/qsort
 36%    74%  138   0.54  0x4004dce0 bsearch->qsort
FUNC      GLOBAL   /lib/libc-2.2.5.so
 20%    94%   78   0.30  0x40098170 memcpy->strsep
FUNC      GLOBAL   /lib/libc-2.2.5.so
  2%    96%    7   0.03  0x4004f5b8 random->srandom_r
FUNC      WEAK     /lib/libc-2.2.5.so
  2%    97%    6   0.02  0x08048524 main
FUNC      GLOBAL   /home/fctype2_alext/dev/linux/prospect/prospect/bench/qsort
 - Skip remainder, SPU of 0.019531 is below '-m 0.020000' Sec

                                                              52,0-1        83%
```

That Wraps it Up

- Prospect's home page is:
  - http://prospect.sourceforge.net
- The project summary page is at:
  - http://sf.net/projects/prospect
- Also of interest is the oprofile site:
  - http://oprofile.sourceforge.net

- Thank you.