2006 linux.conf.au Dunedin, New Zealand

“Steamroller” Testing

Paul E. McKenney
Distinguished Engineer
IBM Linux Technology Center

SMP Code: How to Know When it is Bug-Free?

- Inspection
  - Manual
  - Automated (e.g., sparse)
    - Program analysis vs. model checking
- Testing
  - Functional Testing
  - Stress Testing
  - Dynamic Validation (software, hardware)

- Need all of these – but so what?

Applying RCU to Linux Signal Path

- Signal delivery read-acquires tasklist_lock
  - Degrades latency
- Apply RCU to read-side code!!!
  - Straightforward application of “Reader-Writer-Lock/RCU Analogy” design pattern, very naive
  - Expected failure – but code passed both kernbench and LTP
  - No failure, nothing to debug – but can't be correct
    - Most UNIX® apps “learned” not to trust signal delivery too much!!!
- Oleg Nesterov found some races (good eyes!!!), but still need a good vicious test suite
  - A good test is ***more*** vicious than the users!!!

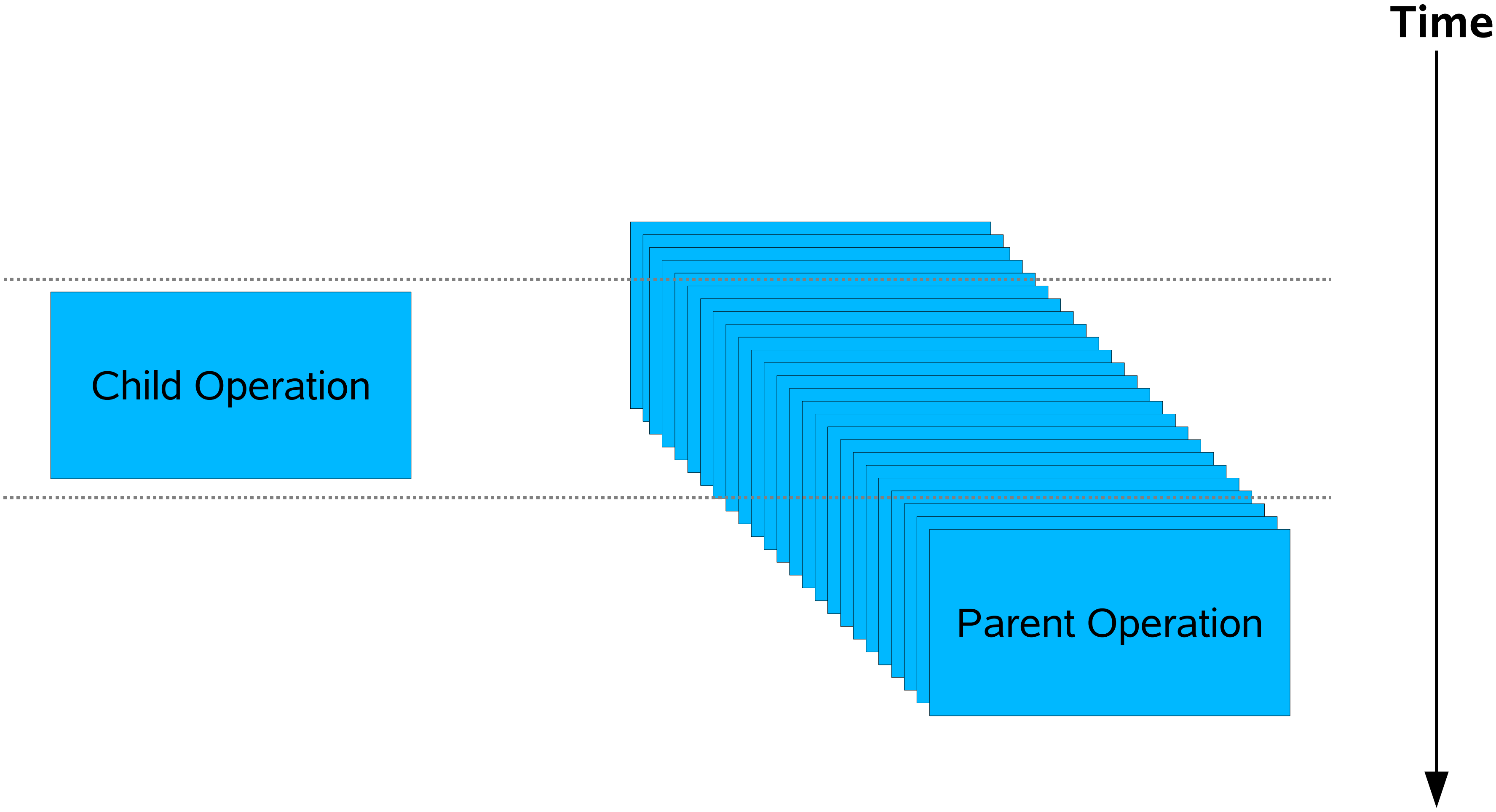
Steamroller Testing: Taking a Leaf from History

- One approach due to Jack Slingwine: ***force*** races to happen!!! Rough pseudocode:

```
for (i = race_begin; i < race_end; i++) {
    retval = fork();
    if (retval == 0) {
        child();
    } else if (retval > 0) {
        for (j = 0; j < i; j++) continue;
        parent();
    } else {
        abort();
    }
}
```

- In theory, forces every possible race to occur...
- How to determine race_begin and race_end?

Steamroller Schematic



Steamroller Testing: Example Output 1

- Testing unicast signal against exit/wait:

```
mckenney@tux1:~/steamroller$ ./sig_exit
steamroller distribution: 73:10600:139
```

- Verbose>=1 prints range:

```
mckenney@tux1:~/steamroller$ ./sig_exit --verbose 1
Race range: 9403:20566 spindelay units
steamroller distribution: 43:10515:605
```

- Verbose>=2 prints progress every five seconds

```
mckenney@tux1:~/steamroller$ ./sig_exit --verbose 2
Race range: 9384:20174 spindelay units
steamroller: 9384 spindelay units
steamroller distribution: 61:10622:107
```

Steamroller Testing: Example Output 2

- Verbose>=2 for killpg vs. fork() storm:

```
mckenney@tux1:~/steamroller$ ./sigpg_forkstorm --verbose 2
Race range: 9561:56529 spindelay units
steamroller: 9561 spindelay units
steamroller: 17929 spindelay units
steamroller: 24842 spindelay units
steamroller: 31947 spindelay units
steamroller: 36858 spindelay units
steamroller: 40666 spindelay units
steamroller: 44745 spindelay units
steamroller: 49702 spindelay units
steamroller: 53958 spindelay units
steamroller distribution: 282:42457:4229
```

- The fork storm self-limits, very useful if you have subtly broken killpg...

Steamroller Testing: Example Output 3

- Verbose>=100 prints exponential and binary-search probes (helps debug new tests):

```
mckenney@tux1:~/steamroller$ ./sig_exit --verbose 100
childdelay = 0  9:0:0
childdelay = 1  9:0:0
childdelay = 2  9:0:0
childdelay = 18205  0:9:0
childdelay = 27308  0:0:9
childdelay = 17699  0:10:0
childdelay = 12894  0:10:0
childdelay = 20390  0:0:10
childdelay = 20355  0:3:7
childdelay = 20372  0:1:9
childdelay = 20363  0:2:8
Race range: 9422:20372 spindelay units
steamroller: 9422 spindelay units
steamroller distribution: 29:10755:166
```

Exponential Search

Binary Search

Steamroller Testing: Complications...

- Need to control what runs on which CPU
  - If parent and child run on same CPU, no race!
- Interrupts, cache effects, &c perturb timings
- All *sorts* of things perturb fork()s timings!!!
- Process vs. pthread primitives

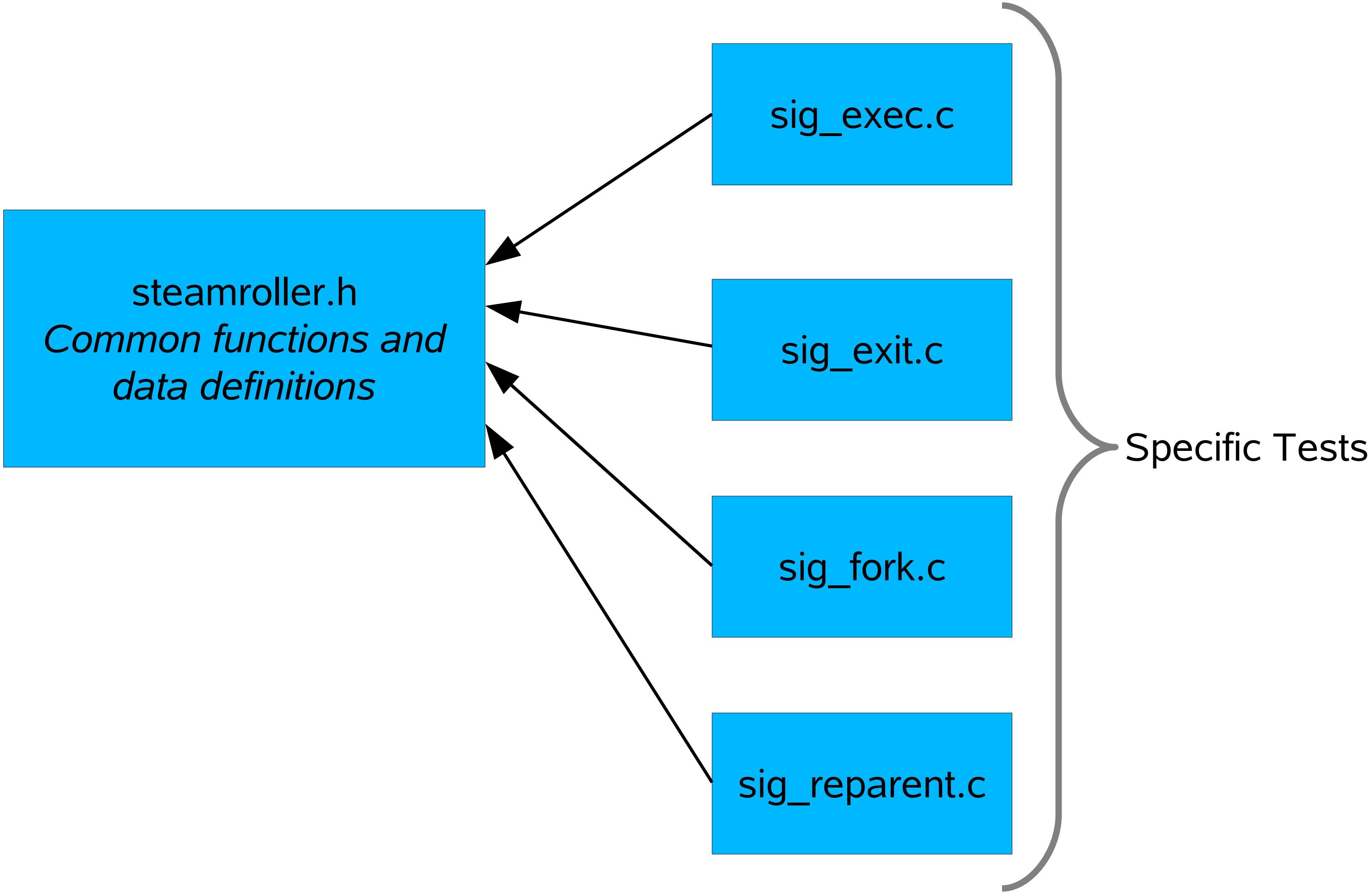
Steamroller Testing: Addressing Complications

- Need to control what runs on which CPU
  - If parent and child run on same CPU, no race!
  - Pass in cpuset to control child, force parent on own CPU
- Interrupts, cache effects, &c perturb timings, and all *sorts* of things perturb fork()s timings!!!
  - Keep system quiet, run multiple times
  - Use smart searching heuristics to locate race
  - Shared variables to synchronize parent and child
- Process vs. pthread primitives
  - Working on this one…

Steamroller Test Creation For Specific Races

How to Create New Steamroller Tests for Specific Races…

Steamroller Program Structure



Steamroller Testing Recipe: Signal vs. exit() main

- Very simple mainprogram:

```
int main(int argc, char *argv[])
{
        long childcpuset;
        void *p;

        childcpuset = steamroller_init(argc, argv);
        p = (int *)mapmem(sizeof(struct sig_test_ctrl), -1);
        search_and_steamroller(test_sig_dfl__exit, p, childcpuset);
}
```

- steamroller_init(): parses args, calibrates spinloop, computes child affinity mask (reserving one CPU for parent), and binding to parent's CPU
- mapmem(): maps memory to be shared between parents and children
- search_and_steamroller(): runs test on specified function (test_sig_dfl__exit()), which must return STEAMROLLER_EARLY, STEAMROLLER_RACED, or STEAMROLLER_LATE

Steamroller Testing Example: Signal vs. exit() test 1

- Definitions and parent-child data structure:

```
#include “steamroller.h”
struct sig_test_ctrl {
        int startflag;
        int raced;
};
```

- Function declaration, local variables, and initialization:

```
int test_sig_dfl__exit(void *p, int parentspin, long childcpuset)
{
        int i;
        int pid;
        int status;
        struct sig_test_ctrl *stp = (struct sig_test_ctrl *)p;

        stp->startflag = 0;
        stp->raced = 0;
```

- General synchronization approach: fork() child, which affinities itself to child cpuset, signals parent via stp->startflag. The parent spins waiting for stp->startflag, then spins for specified parentspin.

Steamroller Testing Example: Signal vs. exit() test 2

- Child code:

```
    if ((pid = fork()) == 0) {
        sched_setaffinity(0, sizeof(childcpuset), &childcpuset);
        stp->startflag = 1;
        spindelay(us2spindelay(100));
        stp->raced = 1;
        _exit(0);
    }
```

- Parent checks for fork() failure, then...

Steamroller Testing Example: Signal vs. exit() test 3

- Parent code:

```
while (stp->startflag == 0) continue;       /* Wait for child to start */
spindelay(parentspin);                      /* Wait for specified spin time */
if (waitpid(pid, &status, WNOHANG) != 0) {
  return STEAMROLLER_LATE;                  /* Child died before we started */
} else {
  if (kill(pid, SIGINT) != 0) {             /* Send racing kill() */
    perror("kill");
    exit(-1);                               /* Should not happen */
  }
  wait(&status);                            /* How did child die? */
  if (WIFEXITED(status)) {
    return STEAMROLLER_RACED;               /* child _exit() won the race */
  } else if (WIFSIGNALED(status)) {
    if (stp->raced == 0) {
      return STEAMROLLER_EARLY;             /* killed child before _exit() */
    } else {
      return STEAMROLLER_RACED;             /* kill() won the race */
    }
  } else {
    fprintf(stderr,
      "strange exit after signal%d\n", status);
    exit(-1);                               /* Should not happen */
  }
}
```

Steamroller Test Debugging: racescan()

```
void racescan(int (*f)(void *, int, long),
              void *p, long childcpuset,
              int start, int mult, int div,
              int lim)
```

- f: streamroller test function
- p: pointer to shared memory for parent-child communications
- childcpuset: CPUs for child. Function f will be called running on the parent's CPU
- start: spinloop count at which to start scan
- mult: multiplier for exponential search
- div: divider for exponential search
- lim: spinloop count at which to stop
- Prints out early/race/late summary for each spinloop count

Steamroller Test Debugging: racescan()

- Common problem: steamroller function never detects race
- Sample output:

```
9:0:0 -- child # 1
9:0:0 -- child # 2
9:0:0 -- child # 4
8:0:1 -- child # 8
5:0:4 -- child # 16
3:0:6 -- child # 32
0:0:9 -- child # 64
1:0:8 -- child # 128
0:0:9 -- child # 256
0:0:9 -- child # 512
```

- Test always reports either STEAMROLLER_EARLY or STEAMROLLER_LATE, never STEAMROLLER_RACED
  - Perhaps due to misinterpretation of error codes or having overly-tight synchronization so that race cannot occur

Steamroller Test Strategy

Steamroller Internals

Steamroller Internals: search_and_steamroller()

- search_and_steamroller() code:

```
void search_and_steamroller(int (*f)(void *, int, long),
          void *p, long childcpuset)
{
  int before;
  int after;
  int resulttab[3];

  if (!racepowersearch(f, p, childcpuset, 0, 3, 2, INT_MAX, 10,
           &before, &after)) {
    fprintf(stderr, "Failed to bracket race.\n");
  }
  steamroller(f, p, childcpuset, before, after, resulttab);
  printf("steamroller distribution: %d:%d:%d\n",
         resulttab[0], resulttab[1], resulttab[2]);
}
```

- racepowersearch() finds the race window using binary search. with initial range from 0 to INT_MAX, using factor-of-1.5 power search
- steamroller() then cycles between the specified bounds
- Printing the number before, during, and after important diagnostic

Steamroller Internals: racepowersearch() 1

- racepowersearch() code initial search:

```
for (i = start; i < lim; i = i * mult / div + 1) {
  raceeval(f, p, childcpuset, i, 9, resulttab);
  if (resulttab[STEAMROLLER_EARLY] >= 7) {
    early = i;
    foundbefore = 1;
  } else if (resulttab[STEAMROLLER_LATE] >= 7) {
    if (foundbefore) {
      late = i;
      foundafter = 1;
      break;
    } else {
      return 0;
    }
  }
}
```

- Return failure if cannot bound the race window
- Note statistical determination of boundary – seven of nine
  - May need more flexibility/configurability longer term

Steamroller Internals: racepowersearch() 2

- racepowersearch() code find lower bound of race window:

```
race = late;
do {
  cur = (early + race) / 2;
  if (racevote(f, p, childcpuset,
               cur, 10, STEAMROLLER_EARLY, 9)) {
    early = cur;
  } else {
    race = cur;
  }
} while (race - early > eps);
*before = early;
```

- Similar code locates the upper bound of race window.
- In this case, racevote() returns true if 9 of 10 evaluations of function “f” return STEAMROLLER_EARLY

Steamroller Testing

Discussion

Steamroller Testing: Experience

- Steamroller produced hangs on RCU signal patches that passed kernbench and LTP
- But straightforward fixes were quite intrusive!!!
  - Maintain per-tasklist lock for state changes
  - Signal delivery acquires lock for thread, process, or process group, depending on scope of signal
- However, was later “inspired” to create a much simpler patch that was clearly correct
- Continuing to use it for testing realtime Linux™ kernels

Steamroller Testing: Limitations & Future Directions 1

- Only tests two operations at a time
  - I have seen races involving up to five operations
  - And probably more that I gave up on!!!
  - But pairs covers a “good and sufficient” set
  - And can always run a background test
- Keep it simple and focused!
  - Works well when testing a small change
  - Test the change against the related operations

Steamroller Testing: Limitations & Future Directions 2

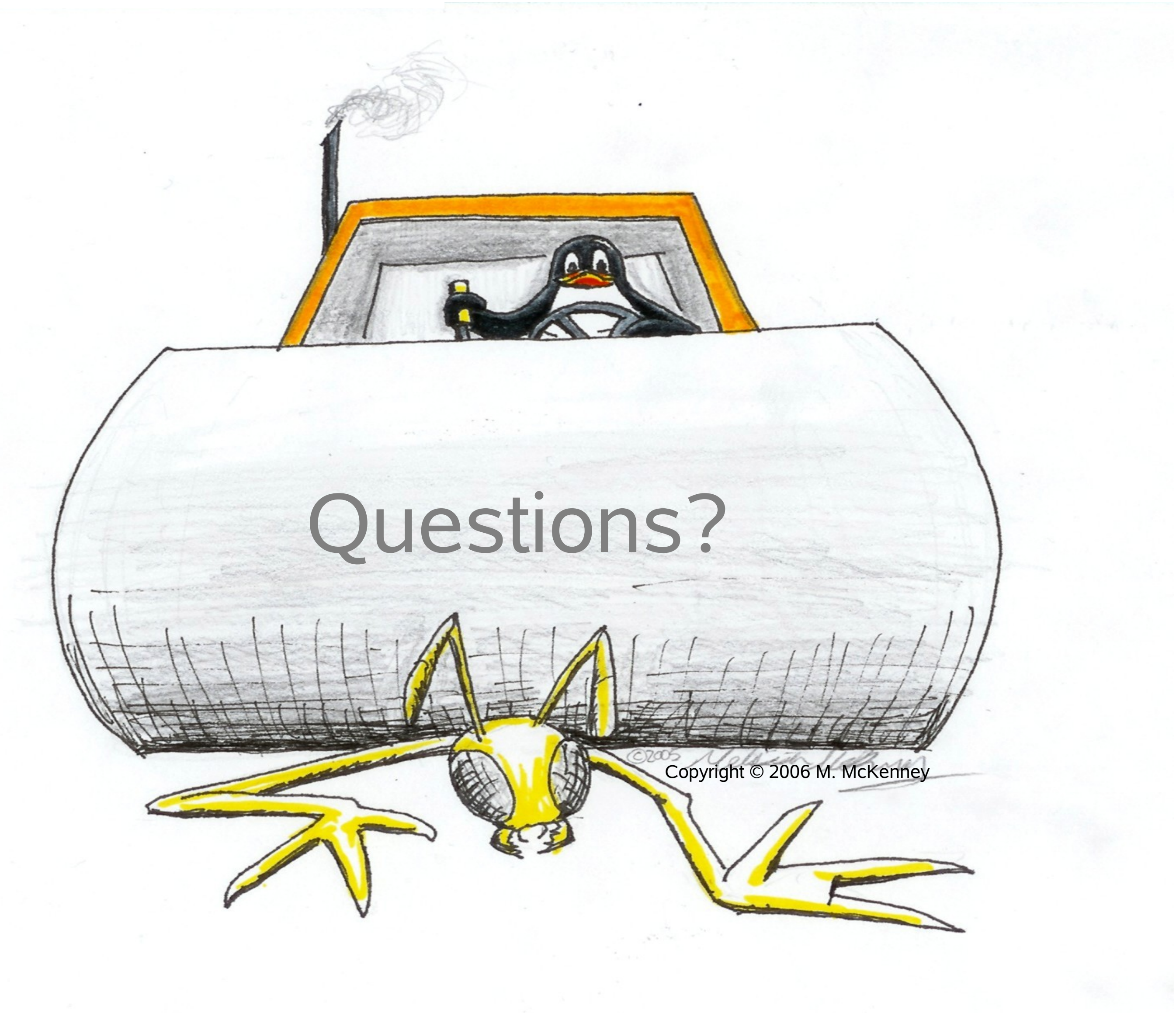
- Combinatorial explosion
  - I was hoping to write single tests and pair them
  - But race detection is quite specific to each pair...
  - Hopefully will come up with a better way...
- State-based races can elude steamroller
- Large race windows results in slow steamrolling
- No explicit pthreads support
  - Can make the steamroller function fork a pthreads child
  - Or add additional pthreads APIs to steamroller.h
  - Hoping for a third option...

Steamroller Testing: Limitations & Future Directions 3

- Hard to detect memory leaks
  - Can use instrumentation if leak suspected, but…
- Tracks down races, but does not necessarily help isolate the actual problem
  - Reproducible test case valuable nonetheless
- Subtle memory corruption caused by a bug exposed by steamroller might take some time to become visible
  - Again, reproducible test case valuable nonetheless

Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
- Other company, product, or service names may be trademarks or service marks of others.



Questions?

Copyright © 2006 M. McKenney

http://www.rdrop.com/users/paulmck/projects/steamroller/