Dynamic Software Updates for C Applications

Sebastian Hahn

Friday 27th June, 2014

Software Update

"There are two ways to write error-free programs; only the third one works."

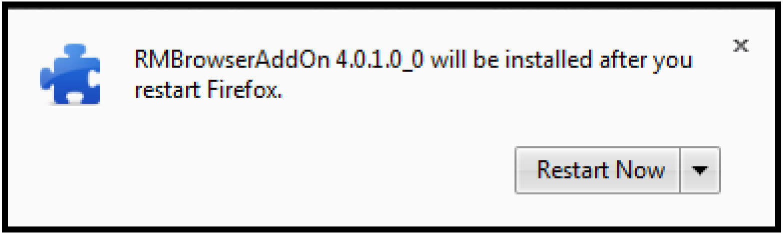
— Alan Perlis

Dealing with the third way

- (Currently accepted) solution: Software updates
- Updating software is easy!

Dealing with the third way

- (Currently accepted) solution: Software updates
- Updating software is easy!



Agenda

Dynamic Software Update for C Server applications

Implementations

- Ginseng
- Stump (Ginseng-MT)
- Kitsune

Results

Goals & Challenges of DSU

- Full state transfer without restart
  - allow updating entire software
  - ... not just small bugfixes
- Updates should be "fast"
  - during normal operation and during updating
  - ... but no realtime requirements
- Assist programmers in generating an update
- Support multithreaded applications
- Robustness against programmer mistakes

DSU tool overview

- Guarantee representation consistency
  - only one version of a function active at any point in time
  - ⇒ restrict updates to points where call stack is short
- Tool-based approaches
  - automatically insert code to take care of the update
  - ease the process of creating patches
  - detect programmer mistakes
- Use of a runtime to manage updates
  - call into runtime to check for updates
  - trigger runtime externally

Agenda

Dynamic Software Update for C Server applications

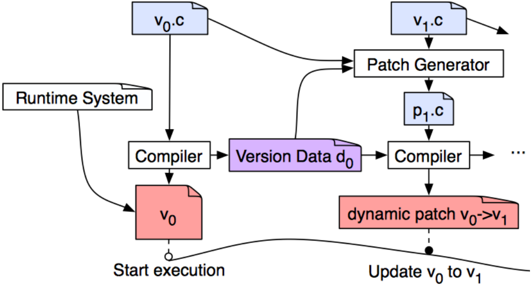
Implementations

- Ginseng
- Stump (Ginseng-MT)
- Kitsune

Results

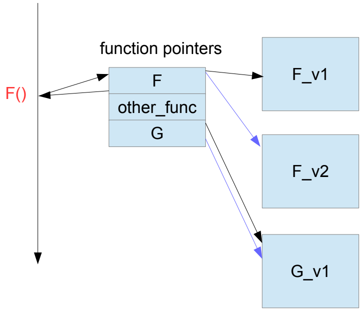
Ginseng

- Supports DSU for single-threaded applications
- Lazy approach to updating
- Published in 2006

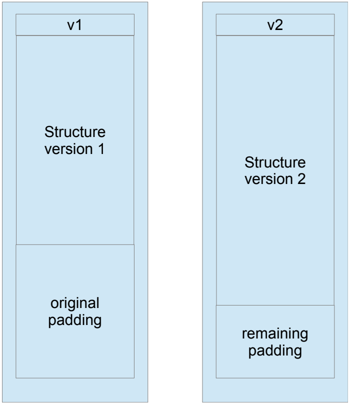


Function indirection & type wrapping

- Function indirection

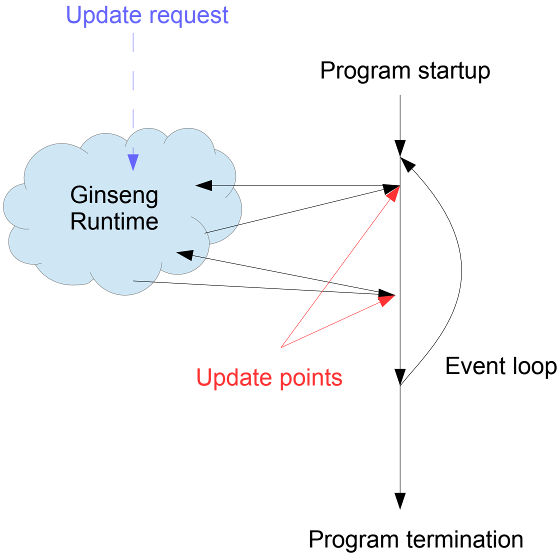


- Type wrapping

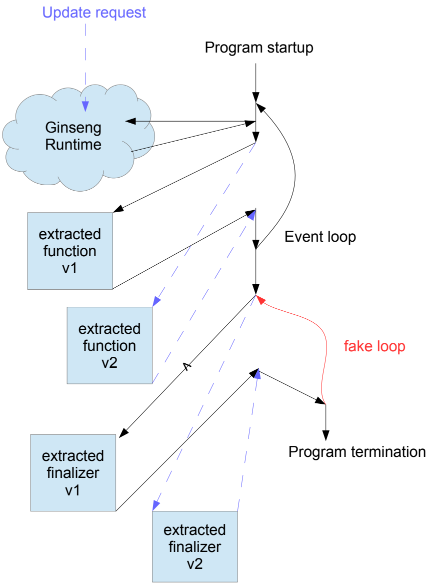


Update points

- User specifies update points
- Safety analysis



Loop extraction



Small example

```
void foo(float g) {
  int x = 2;
  L1: while (1) {
    if (++x == 8)
      break;
  }
}
```

```
struct L1_ls { float *g; int *x; };

int L1_loop(int *ret, struct L1_ls *ls) {
  *(ls->x) = *(ls->x) + 1;
  if (*(ls->x) == 8) return 0;
  else return 1;
}

void foo(float g) {
  int x = 2; int retval; int retcode;
  struct L1_ls ls = { &g, &x };
  while(1) {
    retcode = L1_loop(&retval, &ls);
    if (retcode == 0) break;
    else if (retcode == 1) continue;
    else return (retval);
  }
}
```

Updated applications

- vsftpd - 13 versions (3 years), 25% slowdown
- sshd - 11 versions (3 years), 32% slowdown
- Zebra - 5 versions (4 years), 12% slowdown
- Observations
  - Patch application takes less than 5 ms
  - Memory usage increases during update streak
- Evaluation
  - Ginseng was able to update all tested applications
  - Moderate slowdowns for tested applications
  - Workflow: Add updatability to an application late in development

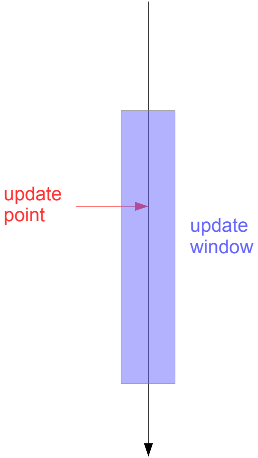
STUMP (Ginseng-MT)

- Same basic architecture as Ginseng
- Improvements for multi-threaded applications
- Published in 2009

Update points

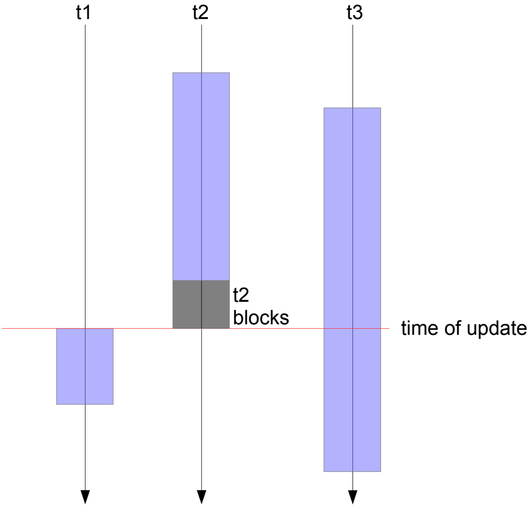
- Simple update points impractical
- Threads block for a long time
- Deadlock potential

- Solution: update windows



Relaxed synchronization

- Check in with runtime
- Wait for all threads



Updated applications

- Icecast - 5 versions, 7% slowdown
- Memcached - 4 versions, 5% slowdown
- Space Tyrant - 7 versions, no slowdown
- Observations & evaluation
  - All tests are performed in an I/O bound state
  - Memory usage increases by 46% for SpaceT
  - Not much has changed compared to Ginseng

Kitsune

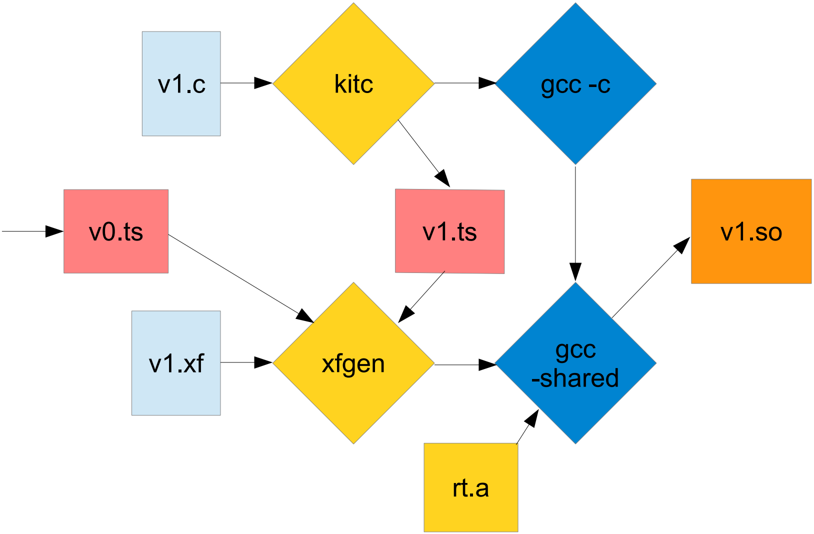
- Whole-program updates
- Borrows from UpStare and Ginseng
- Code publicly available (github) since early 2014
- Published in 2012

KITSUNE

Whole-program updates

- Update entire state at once
- Halt execution until update is complete
- Works seamlessly for many multi-threaded applications
- Higher update complexity
- State conversion
  - programmer has to provide transition functions
  - tools can support the generation of these functions
  - stack reconstruction

Toolchain



Update process

- ## Update preparation
  - Use Unix signals - SIGUSR2 is often unused
  - Block threads as they reach update points
- ## Update execution
  - Once all threads are blocked, link new library
  - Call main function of new code
    - execute update-specific conversion functions
    - reconstruct stack
    - Unload old code & stack
    - hand off execution to specific continuation point

C example

```c
int c foo, c bar, c_size; // config
int *mapping; // array of config options
int main() __attribute__((kitsune_note_locals)) {
  int main_sock, client_sock;
  kitsune_do_automigrate();
  if (!kitsune_is_updating()) {
      load_config();
      mapping = malloc(c_size * 4); }
  if (!MIGRATE_LOCAL(main_sock))
    main_sock = setup_connection();
  while(1) {
    kitsune_update("main"); //call runtime
    client_sock = get_connection(main_sock);
    client_loop(client_sock);
  }
}
```

xfgen example

```
struct list {
  int key; int val; struct list *next;
} *mapping;

mapping -> mapping: {
  int key;
  $out = NULL;
  for (key = 0; key < $oldsym(c_size); key++) {
    if ($in[key] != 0) {
      $newtype(struct list) *cur =
          malloc(sizeof($newtype(struct list)));
      cur->key = key;
      cur->val = $in[key];
      cur->next = $out;
      $out = cur;
} } }
```

Updated applications

- csftpd - 14 versions
- Tor - 13 versions
- redis - 5 versions
- Memcached - 7 versions
- Icecast - 7 versions
- Observations
  - No overhead during non-update usage across the board
  - High memory requirement during update, but freed afterwards
  - Updates can be delayed significantly by sleeping threads

Challenges for updating Tor

- Tor is a networked application
  - connections should not be interrupted by an upgrade
  - large amounts of state for connection handling
- Tor heavily employs cryptography
  - busy relays are CPU-bound
  - crypto mostly implemented in third-party libraries
- Large codebase (76k LoC) with extensive changes
  - still only 159 lines added for Kitsune
  - transformation specification also less than 200 lines
- Tor already uses the SIGUSR2 signal
  - Use existing Tor controller infrastructure

Evaluation

- Kitsune enables DSU without measurable runtime overhead
- Updates are fast even though complete approach is chosen
- Workflow: Integrate DSU as main concern during development

Agenda

Dynamic Software Update for C Server applications

Implementations
- Ginseng
- Stump (Ginseng-MT)
- Kitsune

Results

Discussion of results

- All three tools are effective
- Update streaks possible for all tested applications
- All tools support the programmer in ensuring update safety
- Kitsune is available for user under LGPL
- Kitsune appears to be the most mature and stable tool

Ideas for future work

- Implement updates for Tor spanning multiple release series
- Multi-process applications?
- Updates of NUMA-applications?

Ideas for future work

- Implement updates for Tor spanning multiple release series
- Multi-process applications?
- Updates of NUMA-applications?

Questions?