Tools for finding resource leakages (final)

Sergei Ivanov, SUAI
Supervisor: Eero Tamminen, Maemo devices

Resource leakage

- Memory:

```
...

void test_malloc()
{
        char *x = NULL, *y = NULL;
        x = malloc(123 * sizeof(char));
        y = malloc(456 * sizeof(char));

        free(x);
}


...
```

  - Leakage of size 456 bytes
- Leakages finding:
  - Endurance testing (previous project) – determine only existence of leakages
  - Shows process memory usage from system point of view
  - But couldn't determine place of leakage (source file, line number)
  - New project: changes in different types of resources (not only memory)

Resource types

- Memory
- File descriptors
- GObject
  - Has some similarity with memory
  - Used in GTK
  - May include several types of resources with different sizes
  - In Maemo up to v5 software typically uses GObject as the base object type
- Memcpy
  - Too many memcpy's leads to hi load of memory and CPU
- Threads
  - Used in GTK
  - When joinable thread haven't been joined during application lifetime, kernel cannot release thread related resources (stack etc)

Leakage finding technology

- Program is traced by some specialized tool
- When resource allocation or deallocation occurs, stack of function calls (backtrace) is dumped to the log file
  - Usually backtrace contains pointers to functions and looks like:

```
0. [01:41:53.744754] block at 0x093ef200 with size 24
   /lib/libc.so.6(wcsdup+0x2a) [0xb7e3d78a]
   ./memory_lleaks(test_wcsdup+0x26) [0x80489a4]
   ./memory_lleaks [0x80489ec]
   ./memory_lleaks [0x80489f9]
   ./memory_lleaks [0x8048a13]
   ./memory_lleaks [0x8048a30]
   ./memory_lleaks [0x8048a4d]
```

  - Name resolving is needed:

```
0. [01:41:53.744754] block(24) = 0x093ef200
   0xb7e3d78a: wcsdup+0x2a (c) (in libc-2.5.so)
   0x080489a4: test_wcsdup (memory.c:111 in memory_lleaks)
   0x080489ec: do_backtrace (memory.c:126 in memory_lleaks)
   0x080489f9: x (memory.c:133 in memory_lleaks)
   0x08048a13: h (memory.c:140 in memory_lleaks)
   0x08048a30: g (memory.c:141 in memory_lleaks)
   0x08048a4d: f (memory.c:142 in memory_lleaks)
```

- Log files are rather big
  - Filtrate logs to keep only resource leakages (post filtration)

Existed tools for finding resource leakages

Tool	functracer	latrace	libleaks	mpatrol
Tracked resources	memory, file	Couldn't distinguish type of resources	memory, gobject	memory, memcpy
Name resolving	Extra tool	Built in	Extra tool	Built in
Post filtration	Needed	Needed	Built in	Built in
Rebuilding application source codes	Not needed	Not needed	Needed In newest version not needed	Needed
Shortcomings	Could not trace threads	Logging backtraces only inside dynamic libraries	Too few resource types, sources recompilation	Too few resource types, static linking

- Valgrind
  - Too heavy for mobile device (consumes a lot of system resources)
  - There is no version for arm architecture

Existed tools for finding resource leakages

- Functracer
  - http://repository.maemo.org/pool/maemo4.1.2/free/f/functracer/
- Latrace
  - http://people.redhat.com/jolsa/latrace/index.shtml
- Libleaks
  - Not free yet
  - Planned to be open source
- Mpatrol
  - http://mpatrol.sourceforge.net/

Problems and aims

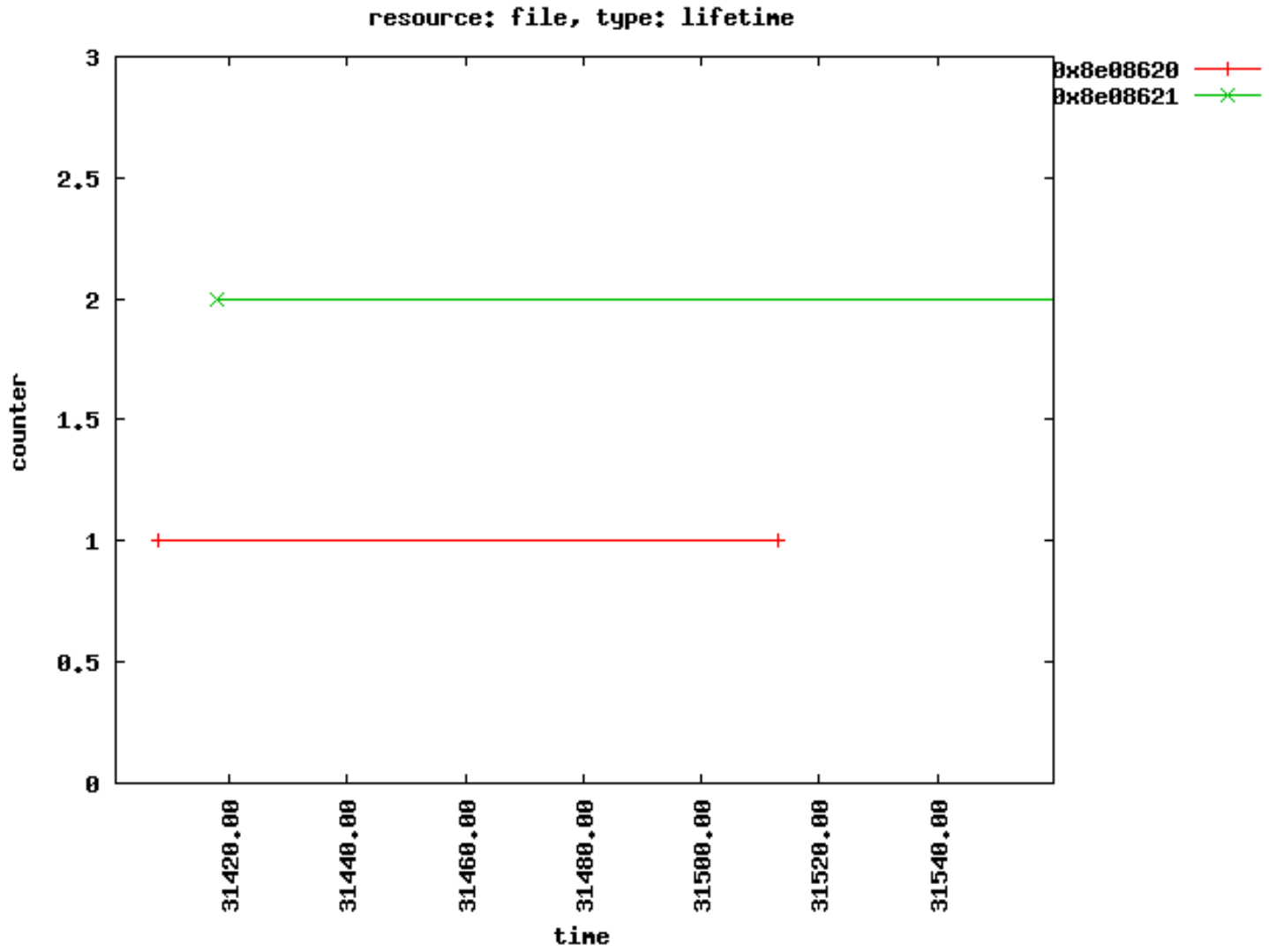
- Not all resource types are supported
  - Extension of existed tools
- Big log files are hard for manual processing
  - Visualization is needed

- Visualization of resource leakages and inefficient resources utilization
  - Are non freed resource allocations existed?
  - What functions have biggest resource leakages size?
  - What is amount of resource leakages during program lifetime?
  - What functions have biggest amount of allocated resources?

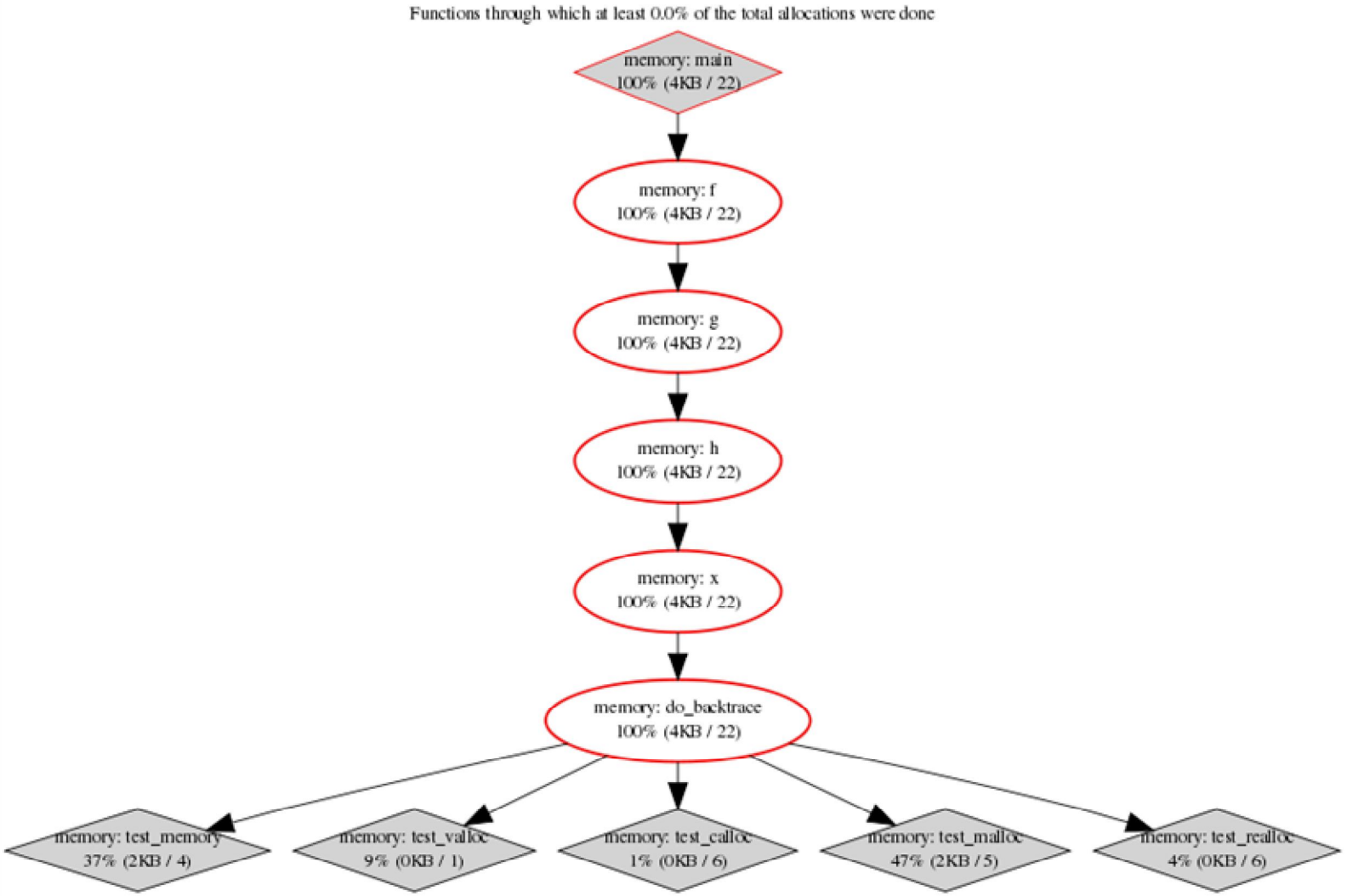
Extending of existing tools

- mpatrol and libleaks haven't any mechanisms of extension
- functracer
  - GObject resource added
  - Memcpy added
  - Improved support of memory and file resources
- latrace
  - GObject
  - File
  - Memcpy
  - Memory
  - Thread

Resource lifetime diagram

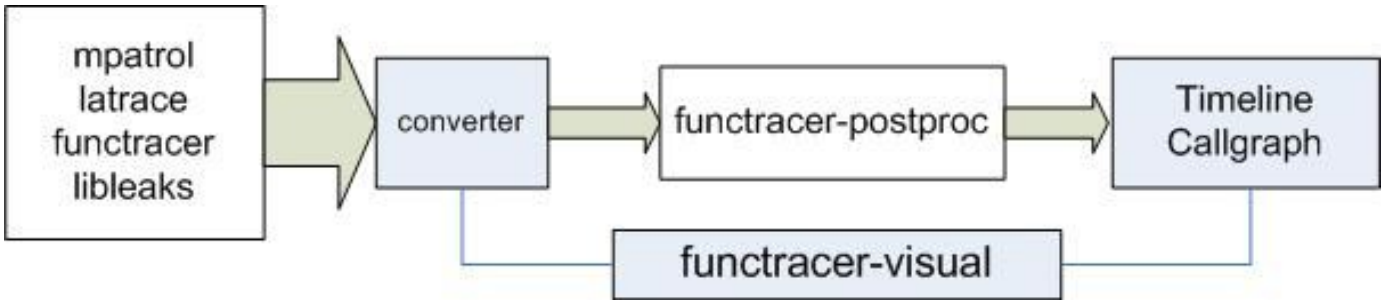


Call tree (backtraces)



Resource leakages visualization

- Different tools have different log files
  - Common log file format is needed



- Functracer-postproc – open source tool for name resolving and finding leakages
- Functracer-visual (developed in the project):
  - Converter – tool for conversion of logs to common format
  - Timeline – tool for visualization of resource lifetimes
  - Callgraph – tool for visualization backtraces to leakages placements

Results

- Some existed tools are extended
  - Functracer
    - Memcpy
    - GObject
    - Improved support of memory and file resources
  - Latrace
    - Differentiation of resource types
- Functracer-visual package:
  - Converter from different logs formats to one common format
  - Tools for visualization of resource leakages
    - Callgraph
    - Timeline

Thanks for attention!

Any questions?