KStruct – Preserving Consistency Through C Annotations

Alexander Schmidt, Martin von Löwis, Andreas Polze

PLOS 2009
Why Do We Monitor?

System comprehension
- When happens what and why?

System administration
- Detect malfunctioning
- Resource shortage
- Detect malware

System debugging
- Analyze bugs
- Improve performance
How Do We Monitor?

**Involuntary Monitoring vs. Voluntary Monitoring**

- **Involuntary Monitoring**
  - Debuggers
  - Instrumentation frameworks

- **Voluntary Monitoring**
  - Performance counters
  - Instrumentation hooks

Running System (Black Box)

Running System (Black Box)
How Do We Monitor?

Involuntary Monitoring vs. Voluntary Monitoring

- Debuggers
- Instrumentation frameworks

- Performance counters
- Instrumentation hooks

Running System (Black Box)
Challenges

Involuntary Monitoring

- Ensure data consistency/quality
- Modelling available data
- Accessing shared state
Data Consistency

Monitored System

LARGE_INTEGER counter;
LOCK lock;
Void Increment()
{
    unsigned int low;
    acquire(lock);
    low = counter.LowPart + 1;
    // check for overflow
    if (low < counter.LowPart)
        counter.HighPart += 1;
    counter.LowPart = low;
    release(lock)
}

Monitor

Void readCounter(PLARGE_INTEGER c)
{
    c->LowPart = counter.LowPart;
    c->HighPart = counter.HighPart;
}
Goals

Data quality

- Leverage (existing) annotations
- Comply to the system’s locking protocol(s)

Flexibility

- Access to arbitrary (shared) heap data

Performance

- Minimize impact on the monitored system
Establish contract between monitored system and monitor on the data object level
- Annotate data structure definition
- Make design intent explicit
- Leverage annotations for inspection

**KStruct Access language**
- Extension to subset of the C programming language
  - struct definitions
- Attribute like syntax
- Extension for defining locking protocols
- Extension for system independent idioms
- Extension for system dependent idioms
Lock annotations
- lock, mlock

Basic type annotations
- cast
- string
- arrays
- root

Idiomatic annotations
- lhead
- fastref
Expressing Locking Semantics

```c
struct EPROCESS {
    KPROCESS Pcb;
    [lock]
    EX_PUSH_LOCK ProcessLock;
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER ExitTime;
    //...
    [mlock(VadRoot)]
    KGUARDED_MUTEX AddressCreationLock;
    //...
    MM_AVL_TABLE VadRoot;
};
```

- Defines the lock of the current container
- Defines the lock of sub-fields
struct EPROCESS {
    KPROCESS Pcb;
    [lock] EX_PUSH_LOCK ProcessLock;
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER ExitTime;
    //...
    [mlock(VadRoot)] KGUARDED_MUTEX AddressCreationLock;
    //...
    MM_AVL_TABLE VadRoot;
};
Architecture

1. C files
   - KStruct Preprocessor
   - Compiler/Linker

2. Rich Symbol Information
   - Setup Phase

3. KStruct Driver
   - Access State Information
   - Running System

4. KStruct Frontend
   - Query
   - Execution Phase

User

PLOS '09 | Alexander Schmidt | KStruct
Object path

- URL like syntax
- First element is a root object
- Further items either
  - Identifier (in lists or arrays)
  - Member of an object
- Object data retrieved by following the object path
CONCLUSIONS
Conclusions

Data quality
- Lock-based annotations
- Idiomatic annotations
- Object paths

Flexibility
- Root annotation
- Structure definition level
Outlook

Refine annotations
- Non-blocking/lock-free locking protocols
- Lock dependencies

Performance

More structure definitions
Questions?

Alexander Schmidt

alexander.schmidt@hpi.uni-potsdam.de
www.dcl.hpi.uni-potsdam.de/research/WRK