Master Thesis – Introductory seminar

Topic: Design and Prototypical Implementation of a High-Level Synchronization Component for Dynamic Updates of Task Run Queues in L4 Fiasco.OC/Genode

Guru Chandrasekhara

Supervisor: Prof. Dr. Uwe Baumgarten

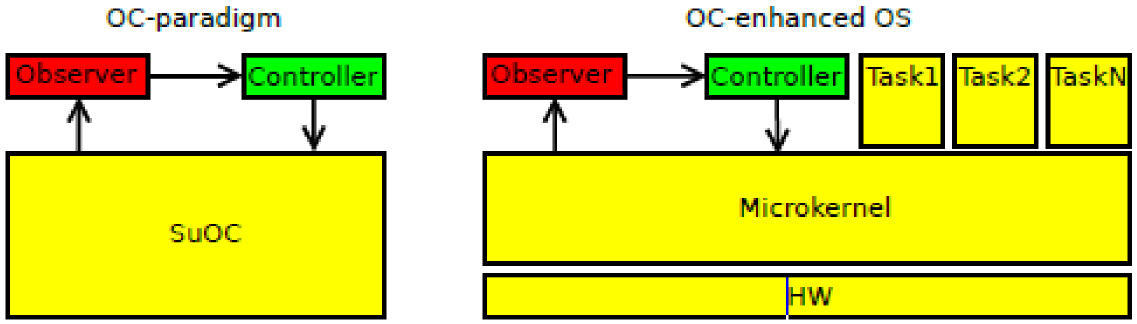
Advisors: Daniel Krefft,M.Sc.; Sebastian Eckl,M.Sc.

Outline

- Motivation
- Problem statement
- System constraints
- Related work
- Comparison/Evaluation of existing methods
- Design
- Implementation/Evaluation

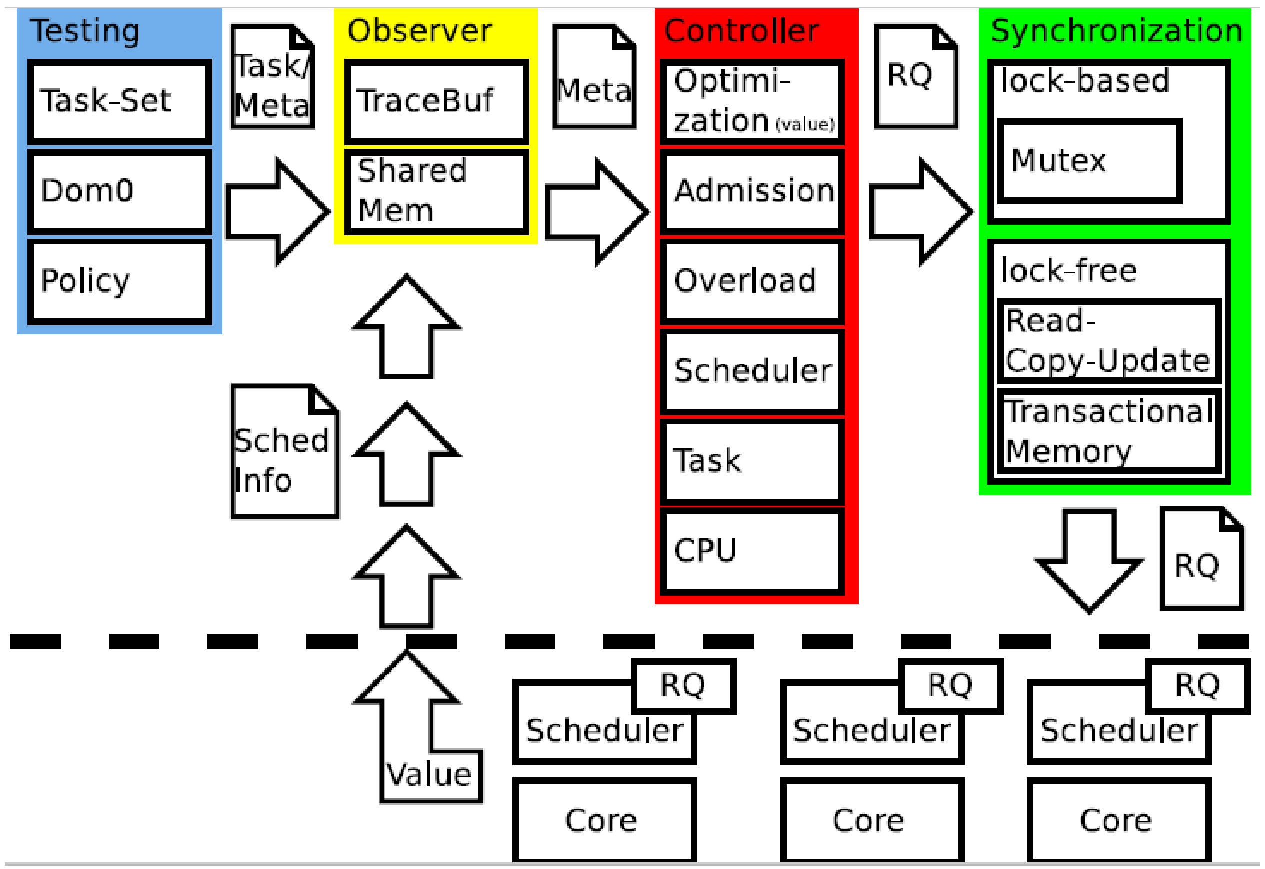
Motivation

- KIA4SM – Cooperative Integration Architecture for Future Smart Mobility Solutions
  - Communication between Intelligent Transport Systems(ITS)
  - Universally applicable ECUs
  - Organic computing
  - Requirements of KIA4SM
    - Flexible scheduling of tasks per ECU
  - Observer controller architecture: Decentralised OC
    - Observers collect data
    - Controllers analyse data



Problem Statement

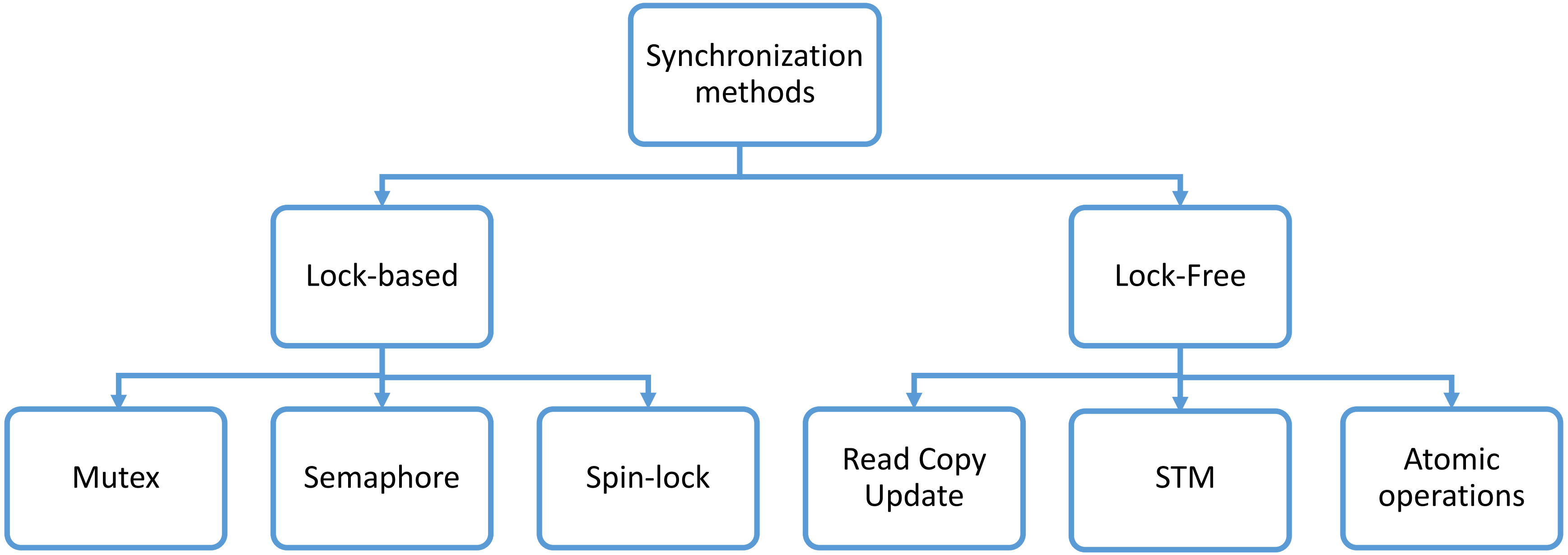
- Controller produces a new Run Queue(RQ)
    - Update the run queue
    - Start executing new run queue
- Producer-Consumer problem
- Synchronization of Run Queue



System constraints

- Real time system requirements
  - Safety – Hard real time system
  - Efficiency – Embedded system
  - Utilization – Threads should reach their deadline
  - Predictable thread synchronization mechanism
- Organic-Computing-Compatibility

Related work: Synchronization Methods



Lock-based: Mutex

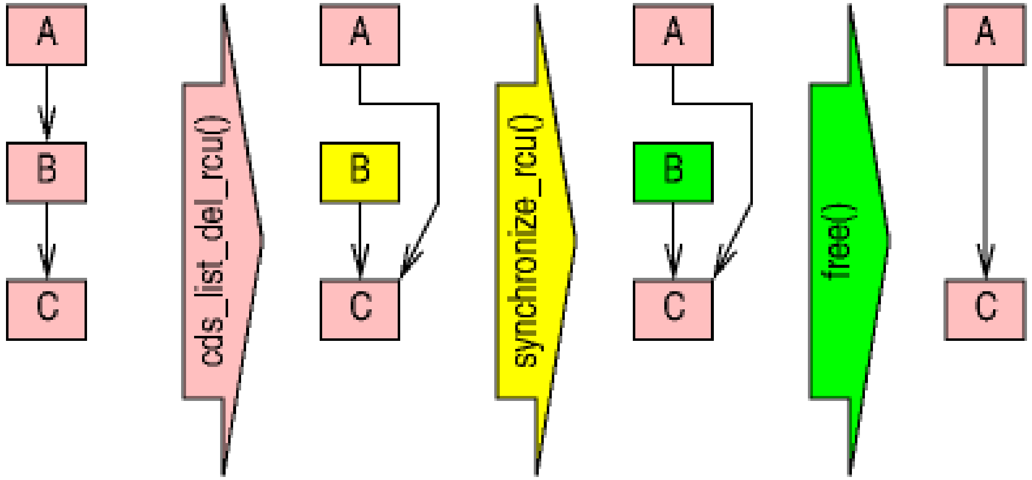
- Simple
- Easy to implement
- Only one thread can access/Modify at a time
- Deadlocks
- Priority inversion
- Starvation

Lock-free: Read Copy Update

- Concurrent read and write operations to the same data
- Three fundamental mechanisms
  - Publish-Subscribe Mechanism (for insertion)
  - Wait For Pre-Existing RCU Readers to Complete (for deletion)
  - Maintain Multiple Versions of Recently Updated Objects (for readers)
- Publish-Subscribe Mechanism
  - Publish: Execute pointer update in right order
  - Subscribe: Fetch the data in right order
    - Memory barrier and compiler directives

Lock-free: Read Copy Update(2)

- Wait For Pre-Existing RCU Readers to Complete
- Deletion of element from linked list



- Grace period

Lock-free: Read Copy Update(3)

- Advantages
  - Concurrent access to shared data structure
  - Low computation and storage overhead
  - Deterministic completion time
- Drawbacks
  - Not a great tool when there are multiple writers
  - Inconsistent data at times
  - Difficult to implement

Lock-Free: Software Transactional Memory

- Concurrency control paradigm that provides atomic and isolated execution for regions of code
- Atomicity, Isolation
- Language extensions are required
  - Data versioning and conflict detection
- Global state of the program is consistent
- GCC 4.7
  - Eg:

```
void testfunc(int *x, int *y) {
        __transaction_atomic {
                *x += *y;
         }
}
```

Lock-Free: Software Transactional Memory(2)

- Advantages
  - Lock-free code is hard to implement
  - More reliable performance
  - Easy to Code (Uses language extensions)
  - No-deadlocks
  - Nested transactions are fine
  - Compiler/Runtime figures out conflicts
- Limitations
  - Running expensive operations multiple times
  - Overhead of transactional monitoring
  - Difficult to debug
  - On fewer cores(<4) performance is not increased

Lock-Free: Atomic operations

- Compare and Swap(CAS)
  - Basic primitive for many lock-free algorithms
  - Handled by CPU instructions
  - Atomically update by comparing memory contents to a given value, if value is same then update

```
int CAS(int *ptr,int oldvalue,int newvalue)
```

  - Perform this operation repeatedly in a loop until it succeeds
- Limitations:
  - ABA problem
  - Considerable overhead (both time and space)

Evaluation of Existing methods

	Mutex	RCU	STM
Implementation	+	--	++
Read-speed	--	++	+
Write-speed	--	+	+
Deadlocks	--	++	+
Overhead	+	++	--
Security/Consistency	++	-	+

Design

- Finding the right point for synchronization
  - The current running thread should execute
  - System should not go into unsafe state
- Empty Run Queue (RQ)

- Static point in time

- Variable point in time

Implementation/Evaluation

- Includes getting the RQ from Controller and Switching it successfully with existing RQ
- Successful execution of new task set/RQ
- Finding the Ideal time for switching the run queue
- Benchmarking different synchronization methods for
  - Speed
  - Safety

Timeline

Apr 15 – May 15	May 15 – Jun 15	Jun 15 – Jul 15	July 15 – Aug 15	Aug 15– Sept 15	Sept15 – Oct 15
Research and Related work	Design	Implementation	Implementation Testing	Testing Benchmarking	Write up

Conclusion

- KIA4SM project
  - Flexible scheduling of tasks per ECU
  - Dynamic thread scheduling necessary to get fault – tolerant system
- Observer-Controller Architecture
  - Produce-Consumer problem
  - Run queue(RQ) synchronization
  - Finding right method and time for synchronization
- State of the Art algorithms
  - Lock-based: Mutex
  - Lock-Free: RCU, STM
- Theoretically RCU gives us the best choice in Lock-Free algorithms

References

- KIA4SM - Sebastian Eckl, Daniel Krefft, Uwe Baumgarten
- Software transactional memory in gcc - Dave boutcher
  - GCC benchmarking with STM, Locks
- What Is RCU? - Paul E. McKenney, IBM Distinguished Engineer
  - RCU fundamentals
- Software Transactional Memory, Siyuan Zhou
  - STM basics, comparison to locks
- A Framework for Implementing Objects and Scheduling Tasks in Lock-Free Real-Time Systems - James H. Anderson and Srikanth Ramamurthy
  - RCU usage in real time systems
- Extending RCU for Realtime and Embedded Workloads: Paul E McKenney