Paper title: Using Read-Copy-Update Techniques for System V IPC in the Linux 2.5 Kernel
Paper authors: Andrea Arcangeli, Mingming Cao, Paul E. McKenney and Dipankar Sarma
Your name: Parth Kansara (115135130)

What problem does the paper address? How does it relate to and improve upon previous work in its domain?
The paper looks at read-copy update (RCU) as a concurrent update mechanism which allows for a read-only access to data structures without the requirement of locking. It has been established to provide better performance as compared to other mechanisms, yet there is not one implementation of RCU that outperforms others. There are trade-offs in latencies and system overheads and none of the existing implementations have surpassed in both.

McK02a showed that there is no best overall algorithm. *rcu-poll* has the shortest latency, while *rcu-ltimer* has the lowest system overhead.

Herlihy93 proposes other methods for getting rid of the lock mechanisms which can be leveraged in combination with some refinements from **Michael02a** and **Michael02b**. However, these still count on expensive atomic operations on the shared storage, which causes pipeline stalls, cache thrashing and memory contention even on read-only accesses.

Gamsa99 and **McK02a** offer an RCU implementation that does not need to suppress preemptions but it results in longer grace periods, which is highly undesirable.

What are the key contributions of the paper?
The paper proposes an RCU-based implementation of the Linux System V primitives, which combines the best features of several RCU implementations. *rcu-poll* uses interrupts to force CPU into a quiescent state, which results in low grace periods but high scheduler overheads. *rcu-ltimer* on the other hand calls the scheduler_tick() function on each CPU with a certain frequency, which has extremely low overheads but a longer grace period. *rcu_sched* uses token-passing, which results in extremely low overheads, while having extremely long grace periods. The paper addresses the high overheads in the *rcu-poll* algorithm and works to eliminate the cache thrashing, but this modified *rcu-poll* is outperformed by *rcu_ltimer.*

Further, the paper explains how the analogy between the reader-writer-lock and RCU was used to replace the global locks in the System V IPC primitives. The global locks were replaced by RCU which guarded the mapping arrays. For guarding the IPC operations, the *per-kern-ipc-perm* locks were used. This modification results in a significant speed up on the database benchmarks.

Briefly describe how the paper's experimental methodology supports the paper's conclusions.
The paper describes in depth the various changes made to the System V's semaphores. Both the *ipc_id* array and the *sem_array* were prefixed with a structure called the *ipc_rcu_kmalloc* containing the *rcu_head* structure. This structure is used by the *call_rcu()* function to track the structures during a grace period. Also, the global *ary* lock is replaced by individual locks for each *sem_array* to guard operations on the corresponding set of semaphores. Lastly, to avoid stale data during the translation of a semaphore ID to *kern_ipc_perm*, a *deleted* flag is set in the *kern_ipc_perm*, whenever a particular *sem_array* is removed.

Semaphore removal uses *ipc_rmid* which sets the *deleted* flag and uses *ipc_rcu_free()* to free up the memory of the semaphore. For acquiring a lock on the semaphore state, the *semop()* system call invokes the *ipc_lock().* Since *ipc_lock()* does not block, the semaphore structure is kept alive for the RCU grace period so that it is not lost before the *ipc_lock()* can check the *deleted* flag. During the grace period, the *call_rcu()* function uses the *rcu_head* structure to queue up the semaphore structure. The *kfree()* function is invoked after the end of the grace period. RCU pushes the expansion of the *ipc_id* array to occur in parallel with the searching of the *ipc_lock()* using the *grow_ary()* function. No *deleted* flag is needed as the old version of the array is kept valid throughout the grace period.

The above modifications were made to the RCU implementation on the System V's semaphores which showed a significant improvement in performance. The system showed an reduction in runtime by an order of magnitude on *semopbench,* a System V semaphore microbenchmark. Also these changes only caused an increase of 5% in the overall lines of code, which is a negligible increase in complexity for the scale of improvement offered in terms of performance.

Write down one question you may want to ask the authors.
Can we eliminate the problem of portability of programs across hardwares with different memory consistency models by conveying the information about synchronization operations at the programming level?