Paper title: FlexSC: Flexible System Call Scheduling with Exception-Less System Calls
Paper authors: Livio Soares, Michael Stumm
Your name: Parth Kansara (115135130)

What problem does the paper address? How does it relate to and improve upon previous work in its domain?
This paper focuses on the drawbacks of synchronous system calls. Generally, the system call invocation involves raising a synchronous exception that surrenders the user-mode execution to a kernel-mode exception handler. The application expects the completion of the system call before continuing execution. These system calls have two kinds of overhead costs associated with them. Direct cost is associated with mode switching where the raised exception flushes the processor pipeline. There is also an indirect cost from the system call pollution of the processor structures which refers to writing of kernel-mode processor state on the L1 caches, TLB and other processor structures. High frequency system call invocation causes user-mode IPC degradation mainly due to the direct cost, whereas for medium frequency, indirect cost is the prime source of this degradation. There is also a negative impact on the kernel-mode IPC due to frequent mode switches.

System call batching: **Cassyopia** assembled independent system calls as a single multi-call. **Xen** and **VMware** also bear this feature. However, multi-calls do not support parallel execution or deal with blocking system calls.

Locality of Execution and Multicores: **Cohort scheduling, Soft timers, Lazy receiver processing** are some of the proposed techniques to improve locality of execution. **Computation Spreading** introduces migration of threads and specialization of cores. **Corey** and **Factored Operating System** proposed core specialization but required micro-kernel OS and were unable to dynamically adapt the core utilization to the workload.

Non-blocking Execution: **Capriccio** proposed improvements to user-level thread libraries for server applications. **Behren** demonstrated efficient management of thread stacks and server performance upgrade using resource aware scheduling. **Asynchronous calls** is a mechanism that implements non-blocking system calls. However, FlexSC differs from these techniques as it calls for complete separation of the invocation and execution of the system calls.

What are the key contributions of the paper?
In lieu of the drawbacks of the synchronous system calls, the authors introduce **exception-less** system calls which improves processor flexibility along with temporal and spatial locality of execution. In this mechanism, whenever a system call occurs, a kernel request is written to a reserved **syscall page**. This system call is then executed asynchronously by **syscall threads** and the result is written back to the syscall page. This method offers flexibility in two ways - batch execution of system calls, which increases temporal locality & provision of executing system calls on a different core, parallel to the user-mode threads, offers spatial locality.

The authors implement a 64-byte syscall page table entry, consisting of a system call number, arguments, status and the result. The syscall threads execute the requests in a syscall page in kernel mode, and are woken up only when the user-space execution is blocked. They can also be scheduled on a different processor core. FlexSC uses two new system calls. **flexsc_register()** is invoked by processes who want to register for using the FlexSC mechanism. Explicit registration avoids initialization overheads for processes that do not use this mechanism and involves two steps - mapping of syscall pages to user-space virtual memory and creation of one syscall thread per syscall page entry. **flexsc_wait()** is invoked when the thread is waiting for a system call execution and cannot progress without it. On execution of this system call, the kernel will put the thread to sleep and wake it up when one of the system calls is executed. The paper also offers FlexSC-Threads, a package for converting synchronous system calls to exception-less calls.

Briefly describe how the paper's experimental methodology supports the paper's conclusions.
The paper measures the overhead of executing an exception-less system call, which is upto 130% faster than a synchronous call on single-core when batching 32 or more calls. In the case of Apache, FlexSC provides an 86% improvement by using batching while an exceptional 116% improvement is seen when performing dynamic core specialization. For MySQl, a throughput improvement of 40% was observed and a BIND DNS Server demonstrated improvements between 30% to 105% depending on the request concurrency. Thus, the implementation of FlexSC was able to prove the effectiveness of the mechanisms proposed by the authors

Write down one question you may want to ask the authors.
The authors have mentioned grouping of system call requests by request type as a possible optimization. How would this improve performance of the exception-less system calls?