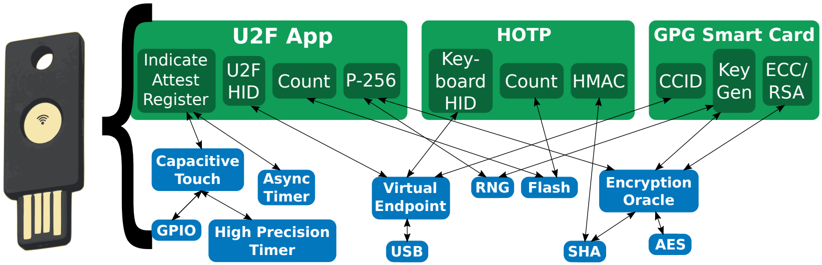
Safely and Efficiently Multiprogramming a 64kB Computer

Amit Levy Daniel Giffin Bradford Campbell Branden Ghena Pat Pannuto Prabal Dutta Philip Levis Niklas Adolfsson Fredrik Nilsson Josh Adkins Neal Jackson
et al. . .

June 14th, 2017



Emerging class of embedded applications are software platforms, rather than single purpose devices.

Embedded Software

- No isolation between components
- Deeply coupled components
- Static memory allocation to avoid unrecoverable runtime memory exhaustion
- Fixed concurrency at compile-time

Embedded Hardware

- Low-power budget—micro-amps average current consumption
- 64kB of RAM
- Memory Protection Unit—a limited hardware protection mechanism

Challenges

- How to isolate components despite minimnal hardware resouces?
- How to replace individual components without restarting the whole system?
- *How to avoid fixed concurrency with limited memory*

Common Solutions

- Give up on isolation—write completely bug-free code

Common Solutions

- Give up on isolation—write completely bug-free code
- Whole system updates only

Common Solutions

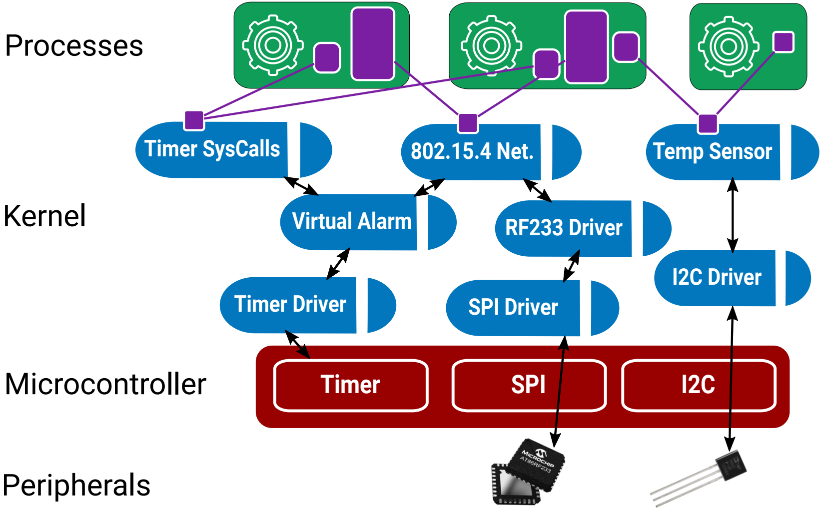
- Give up on isolation—write completely bug-free code
- Whole system updates only
- Use *nix et al—forget about low power

Tock

Tock is a new operating system for low-power platforms that takes advantage of the limited hardware-protection mechanisms available on recent microcontrollers and the type-safety features of the Rust programming language to provide a multiprogramming environment:

- Isolation of software faults
- Efficient memory protection and management for dynamic application workloads
- Update/restart/remove individual (user-space) components independently
- Retains dependability requirements of long-running devices.

Tock Architecutre



Capsules

- Capsules are components in the kernel
- Minimal runtime overhead:
  - Isolated "at compile-time" using the Rust language type/module system
  - Cooperatively scheduled
  - Can eliminate most isolation at compile-time

Capsules can...

- Violate real-time guarantees
- Panic (sort of... lets talk...)

But they cannot...

- Read arbitrary memory (secret encryption keys)
- Communicate with peripherals it's not allowed to

Capsules

Stronger memory isolation than hardware protection?

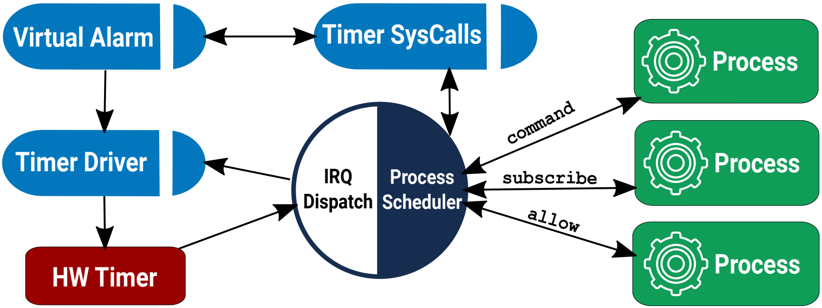
```
struct DMAChannel {
    ...
    enabled: bool,
    buffer: &'static [u8],
}
```

Typing hardware register can constrain allowed values with very fine granularity.

Processes

Can be unreliable since the system can respawn or kill processes without affecting other functionality.

- Hardware isolated concurrent executions of programs
- Written in any language (currently C, C++, Lua and Rust-ish)
- Total control over their memory, including dynamic heap allocation.
- Similar to processes in other systems.
  - Separate stacks allows preemptive execution
  - Memory isolated by the hardware
- Interact with kernel over a small but flexible system-call interface:
  - `command`, `subscribe`, `allow`
  - `yield`, `memop`


Virtual Alarm
Timer SysCalls
Timer Driver
HW Timer
IRQ Dispatch
Process Scheduler
command
subscribe
allow
Process
Process
Process

What happens when the kernel requires dynamic resources to respond to a request from a process?

- We want to allow arbitrary apps so we don't know concurrency requirements:
  - How many timers will an application need?
  - Will it use SPI, UART, USB, Bluetooth, etc? One socket? 1000 sockets?
- If the kernel allocates memory for requests dynamically, it may run out of resources.

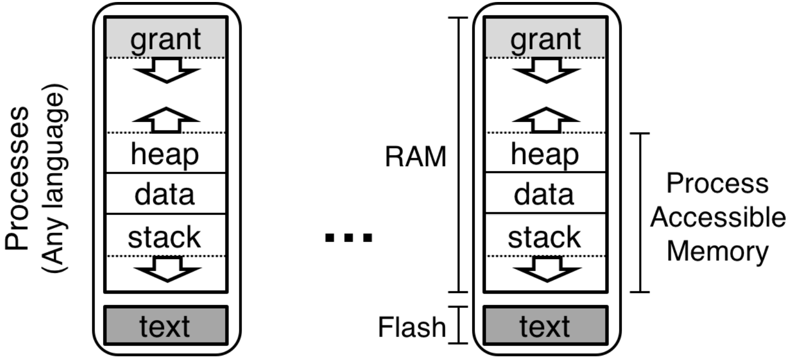
Threads	Kernel RAM	Syscall RAM	Max Used
1	3506	712	158
2	4216	1422	316
3	4928	2134	474

TOSThreads has low memory efficiency. Static allocation costs 710-712 bytes per thread, of which at most 158 bytes (22%) can be in use at any time. These numbers do not include the thread stacks, each of which can be less than 100 bytes.

Grants

Tock allows a process to “grant” to the kernel portions of its own memory, which the kernel can use to maintain state for process requests.

- Separate sections of kernel heap located in each process’s memory space.
- Grant allocations for one process do not affect kernel’s ability to allocate for another.
- Type-safe interface guarantees all grants for a process can be freed immediately if the process dies.
- **Basic idea**: *kernel API ensures there are no long-lived pointers directly to grant-allocated memory.*


Processes
(Any language)
grant
heap
data
stack
text
...
RAM
Flash
grant
heap
data
stack
text
Process
Accessible
Memory

Grant Requirements

- Process cannot access grant allocated memory
  - We use an additional, dynamically determined MPU rule
- Ensure grant-allocated values unavailable to capsules once process dies through limited API:
  - Capsules pass a closure to the `enter` method
  - Memory in a grant region only accessible from within closure
  - Pointers to grant memory cannot escape the closure
  - Implications on kernel design: should avoid cross process data-structures

```
impl<T: Default> Grant {
  fn create() -> Grant<T>

  fn enter<F,R>(&self, proc_id: ProcId, func: F)
    -> Result<R, Error> where
      F: for<'b> FnOnce(&'b mut Owned<T>) -> R, R: Copy

  fn each<F>(&self, func: F) where
    F: for<'b> Fn(&'b mut Owned<T>)
}
```

Grants Compared to the Alternative

Recall: TOSThreads requires 700 bytes staticaly allocated in the kernel for each additional thread. At most 22% can be used at any given time.

- Grants require *no additional per-thread memory* in the kernel
- Only useful memory is dynamically allocated in grants
- *Zero wasted* memory since it can re-use memory for non-concurrent operations.

Conclusion

- Resource constraints continue to be a challenge for embedded system designers.
  - Low-power, small form-factors and lower cost
- These limitations *should not* preclude software abstractions and protections common in general-purpose computers.
- Tock provides both dynamic operation and dependability in resource-constrained settings.
  - Best of all: flexible multiprogramming, isolation, system dependability
- Grants split the kernel heap across processes, allowing dynamic demands for kernel resources despite limited system memory

Buy a Hail! `https://tockos.org/hardware/hail`