**Benefits and Costs of Writing an OS kernel in Rust**

Narek Galstyan under the direction of Prof. Amit Levy

---

### Motivation
- Many of vulnerabilities found in OS kernels could be eliminated had kernels been written in more expressive programming languages.
- Rust is gaining traction in the systems community as a replacement of C.
- IoT ⇒ new Oses needed.
- No direct evaluation of Rust for OS development.

**Linux Kernel Vulnerabilities By Type (1999–2018)**

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denial of Service</td>
<td>1154</td>
</tr>
<tr>
<td>OverRun</td>
<td>340</td>
</tr>
<tr>
<td>SystemSnooping</td>
<td>258</td>
</tr>
<tr>
<td>GainPrivilege</td>
<td>201</td>
</tr>
<tr>
<td>GainInformation</td>
<td>160</td>
</tr>
<tr>
<td>ExecveCall</td>
<td>137</td>
</tr>
<tr>
<td>MemoryCmpArt1</td>
<td>127</td>
</tr>
<tr>
<td>DirectoryTraverse</td>
<td>116</td>
</tr>
</tbody>
</table>

---

### Goals
- Write a kernel in Rust.
- Implement OS kernel in Rust running directly on x86 chips.
- Model kernel components after equivalent components in xv6.
- Pedagogy.
- Evaluate the resulting kernel against xv6.
- Implement Rust and evaluate the resulting kernel.
- Evaluate benefits of Rust and the cost we pay for them.
- Evaluate performance overhead due to Rust.
- Argue about difficulty of using Rust vs C in low level systems.

---

### Performance Cost
- **Rust Provides zero-cost abstractions when possible:**
  - The Rust code above compiles to assembly with no type-checking overhead.

```
fn command_to_drive(ctl: IdeController, port: IdePortArg, value: u8) |
  unsafe {
    outb(ctl as u16 + port as u16, value);
  }
```

---

### Related Work
1. **Singularity:** a Microsoft research Operating system written in Sing#, a flavor of C#.
   - Came before Rust and incurred significant overhead.
2. **Biscuit:** a POSIX compatible kernel written in Go to explore the use of higher level languages in Operating system design.
   - Required a port of Go garbage collector to bare metal.
   - Has significant performance overhead due to GC.
3. **CS140e:** Stanford's experimental OS course taught in Rust.

---

### References

---

### Safe Kernel Scheduler
- **Process schedulers can be arbitrarily complex and workload specific which means they may not be written by kernel developers.**
- **For performance reasons scheduler must run in kernel mode which in C kernels means it can cause arbitrary bugs both in the kernel and inside the processes it orchestrates.**
- In Rust, we can do better:

```
pub fn schedule(pcb: &PCB, N: u32), current: usize => usize {
  pcbs[current].1.iter().enumerate().filter(|(i, e)| e.in_use).
  .map(|e| e.id).
  .next().unwrap_or(current)
}
```

- **Scheduler has read only access to pcb array**
- **Can make scheduling decisions based on process state.**
- **Rust’s memory ownership system will not allow it to modify process state.**
- **Scheduler will still run in kernel mode thereby providing necessary performance but cannot modify arbitrary memory.**

---

### Future Work
- Rewrite parts of existing kernel (Linux or other C kernels) modules in Rust and integrate them into main C codebase.
- Allows to use Rust only for tasks it is best at.
- Enables to directly measure time overhead of using Rust.
- Target hardware architectures other than x86.
- Evaluate the development of more complex and higher level kernel modules in Rust.

---

### Disk IO
- Hardware provides addressable access to permanent storage at 256byte per read/write granularity.
- Need to write a driver that will provide abstract streams to the rest of the system.
- Hardware port IO is inherently unsafe as allows arbitrary data/memory corruption.
- Rust type system allows to expose a constrained safe interface.

```
fn command_to_drive(ctl: IdeController, port: IdePortArg, value: u8) |
  unsafe {
    outb(ctl as u16 + port as u16, value);
  }
```

---

### Should we write Operating Systems in Rust?
- Yes!
- Rust provides many language features such as rich macros, algebraic types, ownership tracking, integrated build system etc. that are quite useful in kernel development.
- But be careful.
  - When writing kernels, sometimes even C is too high level.
  - Additional constraints of Rust may slow you down.
- And choose wisely which parts to write in Rust.
  - Based on my experience in this project, Rust features were more useful in higher level kernel structures. Rust’s type system was very useful when enforcing complex protocol invariants.
  - So, better to write network protocols, filesystems, schedulers of OS in Rust as opposed to lower level kernel modules.

---

### Acknowledgments
- Special thanks to Professor Amit Levy for:
  - Patiently explaining Rust,
  - Helping me navigate x86,
  - And more (in current and next semester).
- Valuable systems research insights.