Evaluation of Rust for Operating System Development and Porting Key Components of the HermitCore Unikernel

Master's Thesis Presentation

Colin Finck

E.ON Energy Research Center | RWTH AACHEN UNIVERSITY

Agenda

Motivation

Motivation

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>

int main(int argc, char* argv[]) {
  char *buf, *filename;
  FILE *fp;
  size_t bytes, len;
  struct stat st;

  switch (argc) {
    case 1:
      printf("Too few arguments!\n");
      return 1;

    case 2:
      filename = argv[argc];
      stat(filename, &st);
      len = st.st_size;

      buf = (char*)malloc(len);
      if (!buf)
        printf("malloc failed!\n", len);
        return 1;

      fp = fopen(filename, "rb");
      bytes = fread(buf, 1, len, fp);
      if (bytes = st.st_size)
        printf("%s", buf);
      else
        printf("fread failed!\n");

    case 3:
      printf("Too many arguments!\n");
      return 1;
  }

  return 0;
}
```

Motivation

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>

int main(int argc, char* argv[]) {
  char *buf, *filename;
  FILE *fp;
  size_t bytes, len;
  struct stat st;

  switch (argc) {
    case 1:
      printf("Too few arguments!\n");
      return 1;

    case 2:
      filename = argv[argc];
      stat(filename, &st);
      len = st.st_size;

      buf = (char*)malloc(len);
      if (!buf)
        printf("malloc failed!\n", len);
        return 1;

      fp = fopen(filename, "rb");
      bytes = fread(buf, 1, len, fp);
      if (bytes = st.st_size)
        printf("%s", buf);
      else
        printf("fread failed!\n");

    case 3:
      printf("Too many arguments!\n");
      return 1;
  }

  return 0;
}
```

Motivation

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>

int main(int argc, char* argv[]) {
 char *buf, *filename;
 FILE *fp;
 size_t bytes, len;
 struct stat st;

 switch (argc) {
  case 1:
   printf("Too few arguments!\n");
   return 1;
```

out-of-bounds access

```c
  case 2:
   filename = argv[argc];
   stat(filename, &st);
   len = st.st_size;
   buf = (char*)malloc(len);
   if (!buf)
    printf("malloc failed!\n", len);
    return 1;

   fp = fopen(filename, "rb");
   bytes = fread(buf, 1, len, fp);
   if (bytes = st.st_size)
    printf("%s", buf);
   else
    printf("fread failed!\n");

  case 3:
   printf("Too many arguments!\n");
   return 1;
 }

 return 0;
}
```

Motivation

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>

int main(int argc, char* argv[]) {
  char *buf, *filename;
  FILE *fp;
  size_t bytes, len;
  struct stat st;

  switch (argc) {
    case 1:
      printf("Too few arguments!\n");
      return 1;

    case 2:
      filename = argv[argc];
      stat(filename, &st);
      len = st.st_size;
```

```c
      buf = (char*)malloc(len);
      if (!buf)
        printf("malloc failed!\n", len);
        return 1;

      fp = fopen(filename, "rb");
      bytes = fread(buf, 1, len, fp);
      if (bytes = st.st_size)
        printf("%s", buf);
      else
        printf("fread failed!\n");

    case 3:
      printf("Too many arguments!\n");
      return 1;
  }

  return 0;
}
```

unchecked return values

Motivation

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>

int main(int argc, char* argv[]) {
  char *buf, *filename;
  FILE *fp;
  size_t bytes, len;
  struct stat st;

  switch (argc) {
    case 1:
      printf("Too few arguments!\n");
      return 1;

    case 2:
      filename = argv[argc];
      stat(filename, &st);
      len = st.st_size;
```

forgotten braces →

```c
      buf = (char*)malloc(len);
      if (!buf)
        printf("malloc failed!\n", len);
        return 1;

      fp = fopen(filename, "rb");
      bytes = fread(buf, 1, len, fp);
      if (bytes = st.st_size)
        printf("%s", buf);
      else
        printf("fread failed!\n");

    case 3:
      printf("Too many arguments!\n");
      return 1;
  }

  return 0;
}
```

Motivation

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>

int main(int argc, char* argv[]) {
  char *buf, *filename;
  FILE *fp;
  size_t bytes, len;
  struct stat st;

  switch (argc) {
    case 1:
      printf("Too few arguments!\n");
      return 1;

    case 2:
      filename = argv[argc];
      stat(filename, &st);
      len = st.st_size;
```

```c
      buf = (char*)malloc(len);
      if (!buf)
        printf("malloc failed!\n", len);
        return 1;

      fp = fopen(filename, "rb");
      bytes = fread(buf, 1, len, fp);
      if (bytes = st.st_size)
        printf("%s", buf);
      else
        printf("fread failed!\n");

    case 3:
      printf("Too many arguments!\n");
      return 1;
  }

  return 0;
}
```

assignment = instead of equality comparison == $\rightarrow$

Motivation

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>

int main(int argc, char* argv[]) {
  char *buf, *filename;
  FILE *fp;
  size_t bytes, len;
  struct stat st;

  switch (argc) {
    case 1:
      printf("Too few arguments!\n");
      return 1;

    case 2:
      filename = argv[argc];
      stat(filename, &st);
      len = st.st_size;
```

```c
      buf = (char*)malloc(len);
      if (!buf)
        printf("malloc failed!\n", len);
        return 1;

      fp = fopen(filename, "rb");
      bytes = fread(buf, 1, len, fp);
      if (bytes = st.st_size)
        printf("%s", buf);
      else
        printf("fread failed!\n");

    case 3:
      printf("Too many arguments!\n");
      return 1;
  }

  return 0;
}
```

buffer overflow due to missing NUL termination →

Motivation

- 29 Lines of Code, with serious bugs in at least 11
  - Assignment = instead of equality comparison ==
  - Buffer overflows
  - File descriptor leak
  - Forgotten braces in multi-line `if`
  - Forgotten `break` in a `switch` statement
  - Forgotten NUL-termination of a string
  - Incorrect argument for format string
  - Memory leak
  - Unchecked cases in a `switch` statement
  - Unchecked return values

Motivation

- 29 Lines of Code, with serious bugs in at least 11
  - Assignment = instead of equality comparison ==
  - Buffer overflows
  - File descriptor leak
  - Forgotten braces in multi-line `if`
  - Forgotten `break` in a `switch` statement
  - Forgotten NUL-termination of a string
  - Incorrect argument for format string
  - Memory leak
  - Unchecked cases in a `switch` statement
  - Unchecked return values

- **But compiles warning-free with the default settings of many C compilers!**

Motivation

- Not an unrealistic scenario, and highly security-relevant!
  - David Wheeler, *The Apple goto fail vulnerability*
    Forgotten braces in multi-line `if`
  - Ed Felten, *The Linux Backdoor Attempt of 2003*
    Assignment = instead of equality comparison ==
  - Paul Ducklin, *The break that broke sudo*
    Forgotten `break` in a `switch` statement
  - Yves Younan, *25 Years of Vulnerabilities*
    Buffer overflows and format string problems among the top security issues

Motivation

Is a 46-year-old programming language still the way to go?

The Rust Programming Language

- Mozilla-sponsored programming language developed since 2006, with emphasis on safety and concurrency
- Competitor to C and C++: Compiled systems language with deterministic memory management
- Implements mature features of C and C++, but also from Haskell, OCaml, SML, etc.
  (no backward compatibility necessary)

The Rust Programming Language

- Mozilla-sponsored programming language developed since 2006, with emphasis on safety and concurrency
- Competitor to C and C++: Compiled systems language with deterministic memory management
- Implements mature features of C and C++, but also from Haskell, OCaml, SML, etc.
  (no backward compatibility necessary)
- **Makes all of the aforementioned code bugs impossible, and many others!**

Some Rust Features

The `str` type

- Combines a buffer and a length
- Guaranteed UTF-8 character encoding
- Bounds-checked at runtime
- Used consistently throughout all of Rust

Some Rust Features

Ownership

1. Each value in Rust has a variable that's called its *owner*.
2. There can only be one owner at a time.
3. When the owner goes out of scope, the value will be dropped.

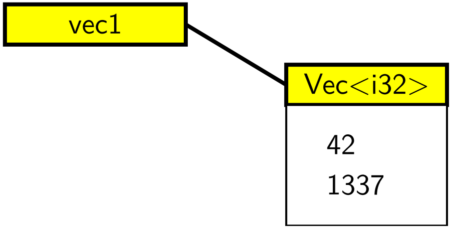
Some Rust Features

Ownership

1. Each value in Rust has a variable that's called its *owner*.
2. There can only be one owner at a time.
3. When the owner goes out of scope, the value will be dropped.

Example

- `let vec1: Vec<i32> = vec![42, 1337];`



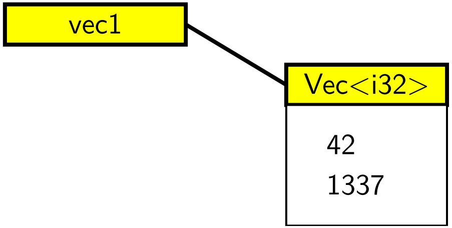
Some Rust Features

Ownership

1. Each value in Rust has a variable that's called its *owner*.
2. There can only be one owner at a time.
3. When the owner goes out of scope, the value will be dropped.

Example

- `let vec1: Vec<i32> = vec![42, 1337];`
- `fn process_vector(input_vec: Vec<i32>)`



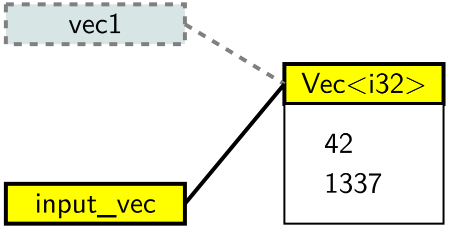
Some Rust Features

Ownership

1. Each value in Rust has a variable that's called its *owner*.
2. There can only be one owner at a time.
3. When the owner goes out of scope, the value will be dropped.

Example

- `let vec1: Vec<i32> = vec![42, 1337];`
- `fn process_vector(input_vec: Vec<i32>)`



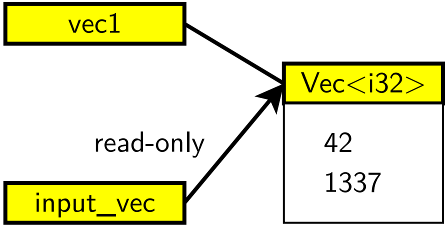
Some Rust Features

References and Borrowing

Some Rust Features

References and Borrowing

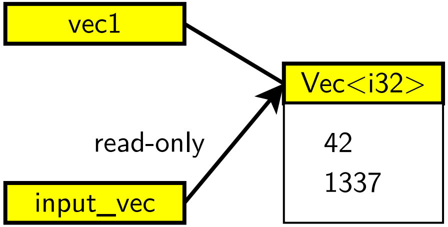
- `fn process_vector(input_vec: &Vec<i32>)`
  - Variable is *borrowed* immutable
  - No transfer of ownership
  - Multiple immutable borrows possible



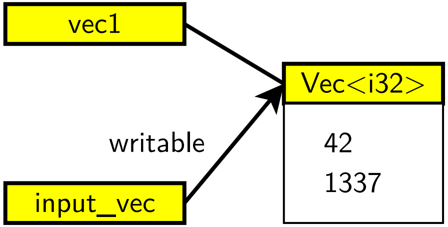
Some Rust Features

References and Borrowing

- `fn process_vector(input_vec: &Vec<i32>)`
  - Variable is *borrowed* immutable
  - No transfer of ownership
  - Multiple immutable borrows possible

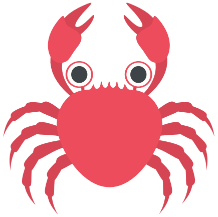


- `fn process_vector(input_vec: &mut Vec<i32>)`
  - Variable is borrowed mutable
  - No transfer of ownership
  - Only one mutable borrow at the same time



The HermitCore Operating System

- Novel operating system kernel developed at the ACS since 2015
- Low system noise and predictable runtime behavior for HPC applications
- Supports GCC (C, C++, Fortran, Go), POSIX, OpenMP, and Pthreads
  - Many existing HPC applications can be easily ported
- Single-address-space library operating system (*Unikernel*)



Thesis Work

Goals

- Porting individual components of HermitCore to Rust while preserving C compatibility
  - Memory Manager
  - x86-64 Hardware Initialization (with APIC and SMP)
  - Scheduler
- Prefer safe and maintainable code over performance during development
- Clean remains of 32-bit x86 specific code in 64-bit x86-64 implementation

Thesis Work

Goals

- Porting individual components of HermitCore to Rust while preserving C compatibility
  - Memory Manager
  - x86-64 Hardware Initialization (with APIC and SMP)
  - Scheduler
- Prefer safe and maintainable code over performance during development
- Clean remains of 32-bit x86 specific code in 64-bit x86-64 implementation

Result

Entire Unikernel mode of HermitCore could be ported within this thesis

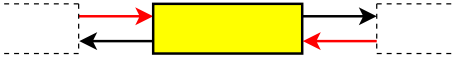
By-Products

- Generic Doubly-Linked List Implementation
  - Not part of the standard Rust library
  - Tricky due to two mutable references per node

Thesis Work

By-Products

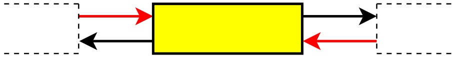
- Generic Doubly-Linked List Implementation
  - Not part of the standard Rust library
  - Tricky due to two mutable references per node



Thesis Work

By-Products

- Generic Doubly-Linked List Implementation
  - Not part of the standard Rust library
  - Tricky due to two mutable references per node



- Generic Free List Implementation
  - Sorted list for managing free blocks of memory
  - Used for both Physical and Virtual Memory Manager
  - Based on Doubly-Linked List

Evaluation

Basic Micro-Benchmarks

System operation	HermitCore Rust	HermitCore C	Linux*
`getpid()`	17	17	143
`sched_yield()`	218	100	370
`malloc()`	764	6080	6575
first write access to a page	27 (4 KiB), 925 (2 MiB)	1407	4007
task switch	5170	934	

in processor cycles

Evaluation

Hourglass Benchmark

	HermitCore Rust	**HermitCore C**	**Linux***
Minimum	24	24	40
Average	30.14	30.15	69.46
Maximum	2551744	5372052	51840

in processor cycles

Evaluation

Code Maintainability

	HermitCore Rust		HermitCore C	
	Files	Lines of Code	Files	Lines of Code
Rust	57	4157	0	0
C Source	8	667	37	5781
C Header	22	866	70	4987
Assembly	6	579	4	932
Sum	**93**	**6269**	**111**	**11700**

Conclusion

- Rust mature enough for operating system development
- Rust increases the productivity
  - Fewer code lines for the same features
  - Compiler catches bugs early
- Added security comes at no significant performance overhead

Conclusion

- Rust mature enough for operating system development
- Rust increases the productivity
  - Fewer code lines for the same features
  - Compiler catches bugs early
- Added security comes at no significant performance overhead

- **If all our software was written in Rust, most security vulnerabilities would be impossible**

Thank you for your kind attention!

Colin Finck – colin.finck@rwth-aachen.de

Institute for Automation of Complex Power Systems
E.ON Energy Research Center, RWTH Aachen University
Mathieustraße 10
52074 Aachen

www.eonerc.rwth-aachen.de

Some Rust Features

The `Result` type

- Can either be `Ok` or `Err`
- Encapsulates the returned data on success or error information otherwise
- Warning if a `Result` type is returned but not checked