The State of the Art

- What is the state of the art in operating systems and systems programming?

The State of the Art

- Most devices run some variant of Unix as their operating system, and are programmed in C
  - Original version of Unix written in assembly for PDP-7 minicomputer in 1969
  - Ported to the PDP-11/40 in early 1970s, re-writing into C at that language was developed
    - “The PDP-11/40 was designed to fit a broad range of applications, from small stand alone situations where the computer consists of only 8K of memory and a processor, to large multi-user, multi-task applications requiring up to 124K of addressable memory space. Among its major features are a fast central processor with a choice of floating point and sophisticated memory management, both of which are hardware options.” https://pdos.csail.mit.edu/6.828/2005/readings/pdp11-40.pdf
  - macOS, iOS, Linux and Android are moderns variants and reimplementations of Unix
- This has proven surprisingly resilient and portable – but is it still the right model?

https://dave.cheney.net/2017/12/04/what-have-we-learned-from-the-pdp-11
Image credit: Dennis Ritchie

Unix and C: Strengths

- Unix gained popularity due to portability and ease of source code access, but also:
  - Small, relatively consistent set of API calls
  - Low-level control
  - Robust and high performance
  - Easy to understand and extend
- Portability was due to the C programming language
  - Simple, easy to understand, easy to port to new architectures
  - Explicit pointers, memory allocation, control of data representation
  - Uniform treatment of memory, device registers, and data structures – easy to write device drivers, network protocols, and interface with external formats
  - Weak type system allows aliasing and sharing

```
struct {
    short errors     : 4;
    short busy       : 1;
    short unit_sel   : 3
    short done       : 1;
    short irq_enable : 1
    short reserved   : 3
    short dev_func   : 2;
    short dev_enable : 1;
} ctrl_reg;

int enable_irq(void)
{
    ctrl_reg *r = 0x80000024;
    ctrl_reg  tmp;

    tmp = *r;
    if (tmp.busy == 0) {
        tmp.irq_enable = 1;
        *r = tmp;
        return 1;
    }
    return 0;
}
```

Example: hardware access in C

Unix and C: Weaknesses

- Unix APIs reflect 1970s/1980s minicomputer architectures
  - Sockets and file system APIs significant performance bottlenecks
  - Security architecture insufficiently flexible
  - No portable APIs for mobility, power management, etc.
  - Assumes professional, interactive, systems administration
- C programming language
  - Limited concurrency support → memory model for pthreads poor supported
  - Undefined behaviour, buffer overflows → security risks
  - Weak type system → difficult to reason about correctness, effectively model problem domain

Unix and C

- Unix has proven surprisingly resilient and portable – but is it still the right model?
  - Maybe – work-arounds for its limitations exist:
    - Kernel bypass networking
    - Increasingly baroque package management
    - Containers and sandboxing
- The C programming language is increasingly a liability
  - Too easy to introduce security vulnerabilities
  - Too easy to trip over undefined behaviour
  - Insufficient abstractions

Some Were Meant for C

The Endurance of an Unmanageable Language

Stephen Kell
Computer Laboratory
University of Cambridge
Cambridge, United Kingdom
stephen.kell@cl.cam.ac.uk

Abstract

The C language leads a double life: as an application programming language of yesteryear, perpetuated by circumstance, and as a systems programming language which remains a weapon of choice decades after its creation. This essay is a C programmer's reaction to the call to abandon ship. It questions several properties commonly held to define the experience of using C; these include unsafety, undefined behaviour, and the motivation of performance. It argues all these are in fact inessential; rather, it traces C's ultimate strength to a *communicative* design which does not fit easily within the usual conception of "a programming language", but can be seen as a counterpoint to so-called "managed languages". This communicativity is what facilitates the essential aspect of system-building: creating parts which interact with other, remote parts—being "alongside" not "within".

CCS Concepts • **Software and its engineering** → **General programming languages**; *Compilers*; • **Social and professional topics** → *History of programming languages*;

Keywords systems programming, virtual machine, managed languages, safety, undefined behavior

ACM Reference Format:
Stephen Kell. 2017. Some Were Meant for C. In *Proceedings of 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Vancouver, Canada, October 25–27, 2017 (Onward!'17)*, 18 pages.
https://doi.org/10.1145/3133850.3133867

Onward!'17, October 25–27, 2017, Vancouver, Canada

ACM ISBN 978-1-4503-5530-8/17/10…$15.00
https://doi.org/10.1145/3133850.3133867

1 Introduction

While some were meant for sea, in tug-boats
'Round the shore's knee,
(Milling with the sand,
and always coming back to land),
For others, up above
Is all they care to think of,
Up there with the birds and clouds, and
Words don't follow.

—Tiny Ruins, from "Priest with Balloons"

I am not ashamed to say that I program in C, and that I enjoy it. This puts me at odds with much of programming language discourse, among both researchers and influential practitioners, which holds that C is evil and must be destroyed. If only we had a "safe systems programming language"! If only we could eke out a little more performance in implementations of other languages, to remove the last remaining motivation for using C! If only we could make "the industry" see the error of its ways! Then C would be eradicated, and there would be much rejoicing.

I am a "systems programmer". It doesn't mean I hack kernels, so much as that I build systems—pieces of infrastructure that integrate multiple interacting parts, and sit underneath application code. Programming in C feels *right* for doing this; it has a viscerally distinctive feeling compared to other, safer, higher-level languages. Certainly, today's experience of programming in C remains, despite certain advances, unforgiving. But I have never felt C to be an encumbrance. C is not a language I use because I'm stuck with it; I use it for positive reasons. This essay explores those reasons and their apparent contrast with conventional wisdom.

2 Two viewpoints

The lyric from which this essay borrows its title evokes two contrasting ways of being: that of the idealist who longs to be among the clouds, and that of the sea-farers who carry on their business on the planet's all-too-limiting surface. The idealist in the song is a priest, who takes literally to the clouds: one day, clutching at helium balloons, he steps off a cliff edge, floats up and away, and

S. Kell, "Some were meant for C: The endurance of an unmanageable language", International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Vancouver, BC, Canada, October 2017. ACM. DOI:10.1145/3133850.3133867

Systems Programming

- What is systems programming?
- **The state of the art**
- Challenges and limitations
- Next steps in systems programming