Interviewer
So I'll start first, just a general question about Rust. What have been your motivations for learning Rust and for choosing to use it?

Participant
I would say I've been mostly using it for personal projects. It's, I think the main motivation is that it puts together the control. Maybe control is not the correct word, but you get the idea. The power, I would say, you have with C++ and speed characteristics and things like that with the ease of use of higher level languages. That would be the main motivation, I would say.

1:1 It's, I thin...
Rust Performs Well

Interviewer
OK. And could you describe a few of the projects that you use it to work on and how Rust is particularly helpful for those?

Participant
OK. Well, I would say I don't actually have that many projects I've written in Rust that I keep working on, but I contributed to a few Rust image decoders where the performance of Rust is pretty much mandatory. And you certainly would, in general, appreciate not risking segmentation faults and other security issues. I wrote like a small… let's call it embedded OS, I'm not sure. Yeah, I completely forgot in the survey that this existed. Oh, never. You know. Basically, a very small operating system that has a purpose to download the image of an operating system, write it on the disk, and then reboot to that machine. It's mostly related to [name]. And it uses the UEFI interfaces, so it's pretty low level. And there is some unsafe hidden here and there that's interesting, I would say.

1:2...
Data Processi...& Ser
1:4 Basic...
Operating & E...dded S

Interviewer
Gotcha

Participant
And the last one, oh, yeah, I actually wrote last week a small server to ask people to compare the difficulties of two problems and just produce a ranking of which problem is hardest and which problem is easiest. And the last thing, I think, is as we have some [name]-related tooling that evaluates a problem and generates inputs and does all of that stuff as a distributed system that is written in Rust and sometimes contribute to that, those, I think, are the biggest projects. And in general, what I said before for Rust applies. Specifically for the embedded project, it's not the first time I write a small embedded project, but it's the first time I write something that's actually usable.

1:7 An...
Personal Tools & Toy P
1:6 And the l...
Networking &...bute

Interviewer
Oh, gotcha.

Participant

And it's definitely like the whole ecosystem around the nostd is miles ahead of anything else that you have in any other language. It's, or at least any other language I work with is, if you wanted to use C++ in an embedded system, you don't have an option. It's like writing C with classes. You don't have the standard library, you have nothing. In Rust, there is an amazing number of crates that you can use without any support whatsoever for embedded in an embedded system without needing an allocator or with your own custom allocator or with other, more or less, with setups.

1:5 And it's definitely like...
Feature Disparity

Interviewer
Gotcha. So that makes it particularly easier to use Rust in that context versus other ...

Participant
Yes, it is a lot easier. And for the small server, I don't know. I just wanted to write a server in Rust and see what happened pretty much. I have to say specifically, one thing I really appreciated is how easy it is to add dependencies that really helped a lot. Of course, the fact that it's fast, there is actually an untrivial amount of computation that goes on in that program. And for example, that disqualifies Python because I want the computation to happen in parallel to the normal execution of the server. And you can't do that in Python because global interpreter lock. And you could write it in C++ if you liked pain. And then I guess it is JavaScript, but it's not my favorite language. So, you know.

1:29 I...
Rust Has Great Toolin

1:8...
C++ is Difficult

Interviewer
Gotcha

Participant
And then there are many other options for server-side programming languages, I would say. So Rust ends up being the best in this situation, at least for me.

Interviewer
Gotcha. So yeah, that's a really broad diversity of applications, but it seems like all of them are particularly suited to using Rust.

Participant
Well, maybe. I may also be, you know, that I am a bit fanboyish and say, oh, yeah, Rust is the best. And so that's why it's best for this use case.

1:3...
Fanboyish

Interviewer
Gotcha

Participant
15 But yeah. So...Specifically around unsafe, I would say that the more

interesting projects are the, well, in the embedded system, I used unsafe, which I guess is not super surprising. And in the image, the coding library, I also used unsafe. As far as I know, at least for my contributions, everything else is completely unsafe free. So specifically for that, there's not much to talk about.

1:3... No Unsafe

Interviewer
Gotcha. So let's focus on those two projects then. When you used unsafe in each of those contexts, what was it for and which unsafe features did you tend to use?

Participant
Let's talk about the embedded OS because it's a more complicated one. So from what I remember, well, there are two things that I used unsafe for. One was to do the equivalent of system calls in a UEFI system. So it's, guess, FFI. Yeah, I guess it's FFI with the UEFI runtime. And the other reason was that I wanted to have like an equivalent of an async executor. As far as I can tell, there is no async executor for, you know, probably there are reasons for that. And I wouldn't have needed unsafe if not for the fact that Rust, of course, tries to be thread safe, but I didn't have locks in, like there is no lock implementation that knows the environment. And for that matter, there aren't threads either. There is only one thread. So basically I mostly used unsafe to tell the compiler, yes, this implements async and it doesn't, but there is also only one thread. So it's okay, you know?

1:9... System Calls

1:41 As far as I can tell... No Other Choice

Interviewer
Yeah

Participant
But that was basically for working around the type system in a situation in which I knew that it couldn't possibly, you know, the guarantees that were required and satisfied wouldn't apply.

1:10 B... No Other Choice

Interviewer
Yeah, sure.

Participant
If that makes sense.

Interviewer
Yeah, yeah. Sort of similar to how you might implement Send or Sync using unsafe on a data structure if you're confident that that follows those rules, right?

31 **Participant**
Yeah, yeah, I mean, I did that. It was, it is not a sound implementation of

... Contract or Invariant

send or sync in a generic environment, but in an environment in which there is only one thread sound by definition because, you know, if there is only one thread, every single implementation of send and sync are sound because they have effectively no information.

1:11 Yeah... Contract or Invariant

Interviewer
Yeah. So when you were calling to the UEFI runtime though, would you need to use unsafe for those calls, right? So were like, which data types did you generally pass to those functions? Was there an interaction between the Rust memory that you had and the UEFI runtimes access to that?

Participant
No, not really. The UEFI run time, well, I mean, there is unsafe UEFI API, which is a memory allocator, but effectively the UEFI APIs are defined in terms of C strings and like, you know, you have 16 bytes.

Interviewer
Gotcha

Participant
Called the function, something happens, the function returns data is no longer used by the UEFI run time, nothing. There is no complex lifetimes involved, effectively.

1:42 C... Simple FFI

Interviewer
So it's with those restrictions, it's fairly easy to ...

Participant
Yeah, yeah.

Interviewer
...safety or is, were there any challenges related to the use of those types?

Participant
No, not at all. I mean, there are already some wrapper functions written around, but, and I use those where I could. In some cases, I couldn't. In some cases, I even just contributed the functions to the upstream implementation, I think. Maybe I didn't contribute them, I [unknown word] the project, I don't know what they are. Something along those lines. Honestly, all the problems I had were with the fact that the UEFI implementations don't actually follow the UEFI specification, but that's a problem that is orthogonal to Rust.

1:12 No,... No Other Choice

1:32 H... Logical Error, Requirements Bug

47 **Interviewer**
Gotcha, gotcha. So the, you mentioned a few different contributions to

upstream APIs. In general, were there cases where, like, are these contributions including unsafe functions that other users of these libraries would call or were they generally safe wrappers around safe functions?

Participant
Generally safe wrappers.

Interviewer
Gotcha

Participant
Around unsafe functions, not. I never exposed an unsafe API to other users.

1:3... Exposing a Safe API

Interviewer
Gotcha. With these APIs, what would it have been necessary to reason about pre and post conditions for calling them or is it generally that with the safe wrapper, as long as you have these safe Rust types, you are fine to call it?

Participant
Well, that's at least what I tried to do. Now that I succeeded at doing it, that's a different question, but in theory, yes. In theory, it should be safe.

1:3... "Safe" API

Interviewer
Gotcha

Participant
It should be safe to call.

Interviewer
Do you have any notion of what types and types of pre and post conditions there could be and whether or not, or how users might be able to violate them or are you fairly confident that your wrappers have helped us?

Participant
I mean, if I had been aware of my wrappers not being a sound, then I would have tried to fix it.

1:1... "Safe" API

Interviewer
Gotcha

Participant
65 Otherwise, I wouldn't have put unsafe on them. Of course, there are always ways you may not be aware of it, but I am fairly sure it should be

1:4... "Safe" API

fine.

"Safe" API

Interviewer
Gotcha

Participant
You know, as sure as you can be.

Interviewer
So you mentioned ...So in the list of types in the survey, you mentioned mostly using slices from your previous response. I'm guessing that was related to the UEFI interface where you're passing C-Strings.

Participant
No. No, that was related to the memory, sorry, the image decoding functions.

Interviewer
Oh, gotcha. Yeah. Let's talk about that then a bit more. Were your uses of unsafe in that project different from the UEFI project?

Participant
Completely.

Interviewer
Gotcha. Could you describe those?

Participant
Those were effectively an implementation of low-level image processing functions. So either discreet design transfer or color conversions. In integer arithmetic and using SIMD operations.

1:14 T...

SIMD Intrinsics

Interviewer
Gotcha

Participant
So the API that the unsafe function exposes is pretty simple.

Interviewer
Mm hmm.

Participant
You get as input two slices. 87

112

113 **Interviewer**

Mm hmm.

Participant
Or maybe, you know, six slices, but you know, input slices and output slices.

Interviewer
Mm hmm.

Participant
And you read values from each slice and modify them and write them to the output slice.

Interviewer
Mm hmm. Gotcha. So you're accessing this memory because you're using slices, you need direct, is it that you just need direct, like, into your indices? And that makes it such that you can't, like, which, I guess, where in particular is unsafe necessary there?

Participant
Well, all the processing is written with SIMD instructions.

Interviewer
Oh, gotcha.

Participant
Like, intrinsics are unsafe.

Interviewer
Mm hmm.

Participant
By default, I would say. Although they are working on something called target feature 1.1, which is ... So basically, the way this works now in the last, so I don't understand this, that functions can have a target feature attribute.

Interviewer
Mm hmm.

Participant
If they have this attribute, it means that they can, this attribute can say, for example, target feature ADX. For example, if they have this function, they can call other functions, this attribute, they can call other functions that have ADX as an attribute.

113

Interviewer
Mm hmm.

Participant
Or, more correctly, the compiler can inline the code from these functions into ...

Interviewer
Yeah

Participant
Intrinsics are implemented as functions that have this target feature attribute. And they are unsafe to call.

Interviewer
Mm hmm.

Participant
Period. The reason why they are unsafe, even if they're just, you know, an addition, is because if you run this on a CPU that doesn't contain, doesn't have ADX instructions, anything could happen.

1:15 T... No Other Choice

Interviewer
Gotcha

Participant
You know, the processor can read that as a different instruction, and who knows what's going to happen, right?

Interviewer
Mm hmm.

Participant
So what the Rust community is working on is this thing called target feature 1.1, which allows, which makes functions with target feature safe to call from other functions with the same target feature.

1:44 S... Shifting Ground

Interviewer
Gotcha So with that particular constraint, like, let's assume that's fully implemented, are there cases where, like, would that make all of your intrinsic usage in that library safe, or are there cases where you would still need unsafe to access memory in a way that is beyond what the borrower checker can do?

159 **Participant**
Oh. Gotcha. That's a good question. So, I think, first of all, when you write, you know, SIMD code, it's mostly about spatial safety as opposed to temporal safety that you need to worry about. There are effectively two kinds of SIMD instructions that would not be immediately, you know, safety trivial. There are load instructions and stores, the same thing, and then gather and scatter instructions. And I think those instructions are different, like fundamentally different, because for a load instruction, what you do effectively is you pass this function appointed and it will load a number of bytes starting from there. So writing a safe wrapper for that would be pretty easy. You know, you just take a slice of a certain size and load the first whatever bytes. And if the slice is too short, that's an error. And you can also make the wrapper for a fixed size slice and all of that. And I guess you need to rely on the compiler to elide the bounds checks at some point. Or you can do some equivalent of ChunksMut, I don't know. But you know, it's not super simple, but it should be relatively simple to make the load instructions and store instructions about as fast as a standard, you know, as what you get, what you would get with unsafe while still being safe. Gather and scatter are different, because gather and scatter just take, you know, for like, I don't know, let's say for indices and for values. So they take four indices and they load four values from memory from those four addresses, or the opposite and like, take the values and write them to memory into those four addresses, and that's not safe. Like, like, like, you can't just bounds check in the beginning and say, well, let's just a slice access. No, the access that you do is not contiguous. So that I suspect would be quite hard to make safe. Or at least I can’t think of a reasonable way to make a safe wrapper around it. At least not a simple way to make a safe wrapper around it. If that makes sense.

1:3… Compiler Elidi...ounds

1:16 Gather and scatter are diff… Difficult to Encapsulat

160

161 **Interviewer**
Yeah.

162

163 **Participant**
All of this, of course, is assuming that targets feature one by one gets merged and everything without that is just all unsafe and the first time. Yeah. Although even with targets feature one by one, you still need them safe once to effectively say, okay, let me check that I have this instruction sets and then let me call the function that uses it. And, you know, I am pretty sure you can write it to a macro of some kind that just generates this code for you and that contains all the unsafe for that for that logic, which is not super complicated logic, right? You just pass it a function that implements some that has some target features and you want to check that that that thing exists and everything. But, but you would need some unsafe to at least start calling, you know, to just say, okay, now I can call the functions that implement this that uses target features.

1:17 Al… No Other Choice

1:18 And,… Macros

164

165 **Interviewer**
Gotcha. So, to confirm, most of the unsafe that you use falls into sort of in this particular area falls into categories where it's either that you just

inherently need unsafe for these intrinsics, or in addition to that, there's also some memory management going on, but that first category, if you have one point one, and you have the instructions that confirmed all those operations would be safe, but in the end, you're still dealing with loads, stores, gather and scatter. And with load and store, you have like if you rely on slices and the length that are built in, you can create a fairly simple safe wrapper for those. But with gather and scatter, it's tricky because you just have four arbitrary memory address. Gotcha. So when you were using gather and scatter, can you describe the situation like which, like where are you taking those addresses from? How do you ensure that calling those are safe in a particular?

Participant
I don't actually use them. I try to avoid the gather and scatter as much as I can because they're other than, you know, being inherently dangerous to use. They are also very slow compared to other instructions. There are alternatives. Like if you know that you need to do a lookup, you need to access things in like short, like a small amount of memory, there are alternatives. And there's just shuffle intrinsics, which have memory trivial, let's say. But I know that they are used in some video codecs.

1:45 I... Preference for Safety

Interviewer
Gotcha.

Participant
Right, because, you know, when you need to look up the data from the previous frame for prediction and things like that, and I think they just do manual bounds checks to ensure that things are fine. And that can fail. It does fail also, there have been memory safety bugs related to this. This was mostly in C++ to be fair, but honestly, when you write intrinsics, it doesn't make that much of a difference if you're writing in C++, C, or Rust. It's like the intrinsics are the same in all the languages and the pitfalls are all the same.

1:19 This wa... Shared Experiences

Interviewer
Gotcha. So let's revisit the UEFI usage just a bit. You're interfacing directly with this runtime. So are you using a existing library that wraps those calls for you? Or are you writing those findings yourself?

Participant
Bit of both.

Interviewer
Okay

165 **Participant**
Basically, I was using an existing library, but it didn't have some of the functions that I needed.

1:4... No Other Choice

Interviewer
Mmm hmm

Participant
I don't remember if I forked the library, sent a PR or something else, but for most things, I use the existing library.

Interviewer
So these are bindings that you just wrote by hand then, without using any tools to assist you?

Participant
Yeah

Interviewer
Gotcha.

Participant
Especially because you just need to extract them from the UEFI specification.

1:4... Generation VS Validati

Interviewer
Gotcha. So, was your process just using the specification and then having that as your guide to write these bindings?

Participant
Yeah.

Interviewer
And were there any unanticipated issues in doing that or was the specification enough to have a fairly straightforward attempt?

Participant
No, because implementers don't actually follow the specification that much.

1:4... Logical Error
Requirements Bug

Interviewer
Okay.

Participant
But that's not a Rust problem, right?

Interviewer
Yeah.

207

208 **Participant**
You know, there are things like, oh, when you receive a packet, this event should fire and that event never fires. And, you know, what do you do at that point?

209

210 **Interviewer**
Yeah. Were, I guess, in writing these bindings or in using them, do Rust developers have to take any considerations of the fact that they can't rely on certain behaviors?

211

212 **Participant**
So what sort of?

213

214 **Interviewer**
So you mentioned that the actual implementation of the UEFI runtime doesn't always follow the specification. Do you feel that, even though that's an orthogonal concern to Rust, do Rust developers have to accommodate that and make any particular decisions that allow that to happen, or is that something that is just unavoidable?

215

216 **Participant**
I would say the situation is not particularly different between Rust or any other language there. It's not that the, except, I guess, there is the, I guess, this cultural thing in Rust libraries that they don't try to hide the complexity from the user. This is a bit more of, this is a meta-observation that I made about Rust developers in general. So I see this tendency, you know, that there are languages in which you try to make the simplest possible API and that has the result, then, you know, there are some complicated aspects that the user then doesn't think about and then they come to bite him or her at some point. Rust has a tendency to, you know, make the API as complicated as it needs to be to fully represent what is actually happening behind the scenes. So one effect of that is that when there are weird things in common UEFI implementations, for example, that tends to be documented in the API, maybe the API tries to have some workarounds around it from the Rust API. I mean, when you make a wrapper API, then you know, you document it, you make a wrapper around it, maybe you decide not to expose some features that are just broken everywhere anyway, and things like that. Or, you know, maybe expose it, but you just put the comment on it and say, this is broken on every single system I tried it on, so please don't use it. But, you know, there is no specific reason why this should be intrinsic to Rust, you know. The exact same thing could happen if one were writing a C++ library, it's just that the culture tends to be a bit different. Another similar thing that I observed related to this cultural aspect. So when I sent some pull requests to this image crate, as I said, I was handling, you know, buffers and slices and checking that the length would be correct and everything. And, you know, I did that by adding some set of asserts at the beginning of the function just to check that everything would be fine. And one thing

1:52 I guess, there is the, I g...
When to Encapsulate?
1:22 I mean, when you m...
Shared Experiences
1:3...
Preference for Safe
1:21 I was...
Runtime Assertion
⋮
Engaging with...st

that somewhat surprised me, like positively, is that the reviewer said, well, this check could be wrong if there is an overflow here. Or some, you know, other similar, like this addition could overflow here, so we should replace this function with a check add instead of a normal addition. Which is completely correct, of course, and I should have thought of that, but the fact that, you know, that the reviewer codes this and noticed this and decided to make sure that this was fine was very welcome. Does that make sense?

1:50 And one thing...

Engaging with...st Com

Preference for Safety

Interviewer
Yeah.

Participant
And I think it also makes, you know, is indicative of the culture of make sure it's correct before anything else that tends to be in Rust, uh, Rust land, let's say.

1:23 A...

Preference for Safety

Interviewer
Gotcha. So to summarize there, there's an emphasis on correctness as a primary concern, as well as in detail of exposing these APIs, such that you are fully documenting the complexities that are involved in using them correctly.

Participant
Yes.

Interviewer
Gotcha. That's really interesting.

Participant
I mean, this is just my observations as a file which

Interviewer
So this is more of a broad question. And I think you've covered parts of it in describing how you use the UEFI interface, but how do you feel that you navigate the differences between rust memory model and the UEFI memory model? Like, are there major differences there or is it released fairly straightforward because you are mostly just passing things with with strings?

Participant
I'm pretty sure the idea, I'm sure there are some parts of the UEFI specification that own the memory, but I haven't found them.

Interviewer
Gotcha.

Participant
So, you know, it's mostly Oh, yeah, pass a buffer in a length and I will fill the data in it and or the other way around. And so that's like pretty memory trivial, pretty trivial as far as memory models are so

1:24 S... Simple FFI

Interviewer
Gotcha. So the next question is, it could relate to any of these libraries that you've worked on, but has there been a particular bug or unexpected behavior in rust that you've written that involves unsafe in some way and how did you find it and address it in development?

Participant
The closest thing that comes to mind to this. Was that there was one specific instruction. I don't remember which one that was intrinsic that had the order of arguments swapped in some version of rust. You know, when they were still nightly so. Before I want to stable. And I don't remember how I mean, the temporary fixes to just swap the arguments. And I think I ended up, you know, using a CFG checks to check whether this is newer or older than that, that specific rust version and do one thing or the other or maybe I just ended up implement using a different intrinsic entirely. But I, you know, this is the kind of thing that can happen. But I haven't found and it's also reasonably easy to figure out that it happened, because, you know, doing one minus two and getting one instead of minus one or some, some similar behavior. And I can't recall any particularly hard to find or particularly problematic unsafe related bugs other than this one.

1:25 Was... FFI & Binding Bug
Mismatched B...r Decla

Interviewer
Gotcha. So it was more just differences in the API that were then fixed and not something that would you would have encountered if it was written correctly.

Participant
No, no not at all.

Interviewer
And then do you feel that there are any challenges that you face either like in terms of correctness or just in terms of the workflows and expectations you've described with unsafe that that the tools you use can't handle well or that there could be a better solution for

Participant
Nothing what I did, but I also, you know, never touched anything that stood pointers, for example, or references like, you know, if I had to work on a slice of references to something and had to ensure that I wasn't violating the borrow check that the uniqueness constraints. Then my answer may be made different, different. But in terms of what I've been doing so far, all pretty simple. Honestly, for what I've been doing the same

1:51 Nothing wh... It's not a problem
⋮ Shared Experience

tools that work for C++ also work for us. Just compile it with the memory sanitizer and the address sanitizer and hope that it works. For the embedded system, the embedded OS thing. Well, that's, that's a bit trickier, right, because there is no address sanitizer for that, I think. So, yeah, I don't know. I don't recall it being useful, like it was, you know, sufficiently simple and safe coded. I wouldn't, I don't think I would have needed. You know, particular tooling for it to debug stuff with that. But, you know, I'm not excluding that it could be really useful in some situations to have some, you know, some tooling support that doesn't require specific runtime, for example.

1:3... Dynamic Analysis Saf

Interviewer
Yeah, yeah, that makes sense.

Participant
Or, you know, maybe something that requires runtime, but you can specify, you know, I don't know. I don't remember exactly language items, I think is how they're coded in Rust. You know, there are two or three functions that you can just tell the compiler. This is the implementation of stack unwinding or whatever. And, you know, if you write a version of address sanitizer that when it crashes, it calls a specific function and that could be extremely useful, I guess, if the runtime that it needs is not super big.

Interviewer
Gotcha. So just extending it to address those particular features that you find in those code bases.

Participant
Yeah, I mean, I don't know, I think it would be really good to be able to have an embedded version of address sanitizer, but never, I didn't need to myself.

1:26 Yea... Wants Solution

Interviewer
Gotcha. Gotcha. But if you were to use some more borrow checker heavy related features in that context, you would anticipate that being more of a useful thing.

Participant
Yes, maybe not an address sanitizer. Maybe a unique, uh, pointer sanitizer. But whatever, you know, some sanitizer.

1:2... Wants Solution

248

262

Yeah. I suspect that that could be very useful. On the other hand, like there is this tendency in Rust to keep your uses of unsafe as local as possible.

1:27 Yeah... Preference for Safety

263

Interviewer
Mm hmm.

Participant
You know, just write one unsafe module that wraps everything around. Which is, I think a development practice that, which reduces the need for such sanitizers.

1:39 Y...
Preference for Safety

Interviewer
Yeah

Participant
Although it doesn't completely remove it, right? It just ... it would still be useful, I think.

Interviewer
Yeah

Participant
Not as useful as it would be in C++, let's put it this way.