Pyromaniac: Evolution

Adapt, Evolve or Die

Gerph, November 2025

Pyromaniac: Evolution

Adapt, Evolve or Die

Gerph, November 2025

0 / 51

0. Introduction

0. Introduction

Good evening. It's been a few years since I talked about RISC OS Pyromaniac here, so things have changed a lot.

1 / 51

Introduction

What I'll talk about

I'll be talking about ...

1. Recap on RISC OS Pyromaniac.
2. Towards 64-bit RISC OS.
3. Changing my approach.
4. Conclusion.

Hopefully I won't be *too* technical, but there will be examples of running code.

Introduction

What I'll talk about

What will I talk about?
I'm going to talk about how the project has evolved over the years, from being just a way to test things to providing new ways to develop and test software. I've split this up into a few sections, and there will be a short break between each to talk about the topics I've discussed.
Please feel free to use the chat to ask questions as we go. I will try to pay attention to it and answer anything that comes up as we go.
At the end of the presentation, these slides and a bunch of links to resources will be made available, and I'll take questions.

2 / 51

Introduction

Who am I?

- I'm Charles, but known as Gerph in most things online.
- I worked at RISCOS Ltd and produced RISC OS Select.
- I ported almost all of RISC OS to be 32 bit.
- I've written the only other implementation of RISC OS - RISC OS Pyromaniac - from scratch.
- I'm a strong advocate of taking RISC OS forward, rather than letting it stay stagnant.

Introduction

Who am I?

Let me introduce myself. I'm Charles, previously known as Justin, better known as Gerph. I've done work on RISC OS since the early 90s, patching and porting things, developing RISC OS itself, and most recently creating a new implementation of the OS itself - RISC OS Pyromaniac.
I've always been a strong advocate of moving RISC OS forwards, and the work that I've done over the last few years is a testament to that.

3 / 51

1. Recap on RISC OS Pyromaniac

1. Recap on RISC OS Pyromaniac

It's been a long time since I talked specifically about RISC OS Pyromaniac.

4 / 51

Recap on RISC OS Pyromaniac

What is RISC OS Pyromaniac? (1)

- An implementation of RISC OS.
- Intended for testing, development, debugging and experimentation.
- A system that helps you to try things out.
- Runs on modern systems.
- Written in high level language.

Recap on RISC OS Pyromaniac

What is RISC OS Pyromaniac? (1)

So what is it?
RISC OS Pyromaniac is intended for testing, debugging, developing and experimenting with RISC OS interfaces. It's not intended for everyday use, but to make the process of developing and prototyping easier.
It's written in the high level language Python. It runs at the command line and within a window on Windows, Linux and macOS.

5 / 51

Recap on RISC OS Pyromaniac

What is RISC OS Pyromaniac? (2)

- Easy to run things without hardware or system emulators.
- Provides many debug options to explain what is happening in the system.
- Allows a full disassembly of every instruction.
- High level language means changing behaviour is easier.
- Cloud systems mean that automated testing is possible.
- Use the advanced editors, IDEs and AI to develop your code.

Recap on RISC OS Pyromaniac

What is RISC OS Pyromaniac? (2)

What does that actually mean?
Well, it means that if you want to write some code and see whether it works, you can do that easily without having to use the RISC OS Classic system. No RPCEmu involved. No physical hardware. You can just run your program and see how it works.
If you want to debug what it's doing, you can easily enable many different options to make it easier to see what's happening. If you want, you can get a disassembly of every instruction that's executed, or trap accesses to memory, or calls to any functions.
That makes it easier to see what's being done and what's going wrong.
If you're considering changing how a part of RISC OS works, you can modify the interfaces to make it work differently. Because RISC OS Pyromaniac is implemented in Python, changing the behaviour is easy. No messing with assembler, rebuilding modules or ROMs, or rebooting.
Because it runs on Windows, Linux or macOS, you can do all this on any modern system you have access to. Or, use the cloud-based build service to test your code automatically every time you make changes.
With RISC OS Pyromaniac, trying things out and ensuring that you have reliable software becomes much easier. Use the tools that you are familiar with - your editors, source control, even A.I. tools if you like - on the platform you are most familiar with. It might not be the most up to date of systems but it allows proper engineering practices to be applied to software development.

Recap on RISC OS Pyromaniac

What's under the hood? (1)

- RISC OS Pyromaniac implements documented RISC OS interfaces.
- Doesn't implement everything - depends on what you're testing.
- Doesn't have hardware - no memory mapped devices.
- Does implement RISC OS interfaces to drivers.
- OS is written in Python.
- Emulation provided by Unicorn - a QEmu-derived system.

Recap on RISC OS Pyromaniac

What's under the hood? (1)

Occasionally I am asked, what version of RISC OS the Pyromaniac system implements. The answer is varied - it depends on what you want. The system has more capabilities and integration with modern systems than most versions of RISC OS Classic. It implements some Select features, some RISC OS 5 features. It absolutely depends on what you're trying to test.

If some feature isn't working as expected, or you want to try it differently, then - as mentioned - it's Python, so it's easy to change.

Similarly, there are often comments about what hardware it provides. The answer there is simple. It does not provide any emulation of hardware whatsoever - other than the CPU, and even that is generic. The point is that hardware drivers are the domain of physical hardware and need to be implemented for such systems. If you're writing a hardware driver, you know what you're working to. And integrating with RISC OS is generally much simpler than getting the hardware to work as expected.

Like WINE, RISC OS Pyromaniac is not an emulator. It includes an emulator for the ARM (or AArch64) components, but the operating system is in Python, translating what you're doing to the host system. Or if you want to think of it in more Acorn-like terms, it is similar to 65Host, interfacing the operating system calls to the host system. Or the BBC Micro's Tube, where the system calls are passed to the host system. Or the ARM semi-hosting environment. There are many examples that are similar to the way in which RISC OS Pyromaniac functions.

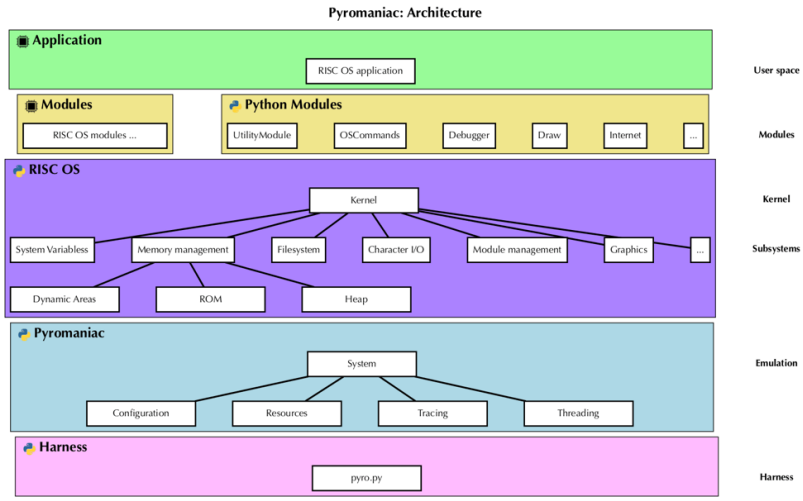
RISC OS Pyromaniac only provides RISC OS APIs, not hardware access. This means that if you write RISC OS code that doesn't access hardware you will be able to use your code on RISC OS Pyromaniac. Which means just about all software - as only a tiny proportion has any hardware specific component.

It's written in Python, as I mentioned, so integrating other well known tools into the system isn't too hard. The back end is Unicorn - a QEmu-derived emulation system. Unicorn provides the ARM emulation that the system uses. QEmu is one of the most well known platforms, and Unicorn makes the interfaces available programmatically.

7 / 51

Recap on RISC OS Pyromaniac

What's under the hood? (2)



Recap on RISC OS Pyromaniac

What's under the hood? (2)

This is a simple diagram showing how parts of the system are layered. The emulated code lives up in user space, and can run in modules. Some modules are in Python. The Kernel and the bulk of the interfaces are in Python. The emulation system - Pyromaniac - is below that, and finally there is the `pyro` tool that we use to run it.

8 / 51

Recap on RISC OS Pyromaniac

Why do this?

- I enjoy it.
- I want to make it easier to develop software.
- I want to enable others to do things.
- I want to be able to use modern techniques for development.

... and from that, it's grown far beyond my original goals.

Recap on RISC OS Pyromaniac

Why do this?

Why did I want to do this?
I've been working on RISC OS Pyromaniac for over 6 years now. Seriously. One release created per month and November 2025 is release 77. That's a lot of time working on one project. Yes, I enjoy it.
But there is a point beyond just enjoying it. I have always wanted to build things that let other people do things. Operating System development is clearly a big part of enabling others to do things. Initially, I wanted a way to test out my changes to the compiler and modules that I was playing with.
RISC OS Pyromaniac grew and now... well; now it can run games. It powers the online build service. It can run the desktop. It has integration with the host system. And it is extremely in debugging things. It has met my technical needs, and so I'm now using it for those original goals - as a way of testing things and experimenting with RISC OS.

9 / 51

RISC OS Pyromaniac demo

RISC OS Pyromaniac demo

For anyone who hasn't seen RISC OS Pyromaniac running before... I'll do a very quick demo.

2. Towards 64-bit RISC OS

2. Towards 64-bit RISC OS

11 / 51

Towards 64-bit RISC OS

What's changed? (1)

Lots of things!
The detail is in the Pyromaniac changelog on the website.
Most of it relates to 64-bit work.

Towards 64-bit RISC OS

What's changed? (1)

What's changed ?
So what's changed with RISC OS Pyromaniac since I presented here?
Lots.
So much so that I'm really not going to go through the changes. It's all described in the change log on the website if you're interested, but the system is a lot more featureful and more robust. It's still not complete, but that's not surprising - RISC OS is a large system.

Towards 64-bit RISC OS

What's changed? (2)

Towards 64-bit RISC OS

What's changed? (2)

Since 2022...

- I've implemented a chunk of the Window Manager rendering implemented.
- The command server allows you to interact with RISC OS at the host shell.
- The graphics support is much improved due to the games.
- Teletext modes work.

And many many more things.
A better question to ask is 'What's driven the changes?'

Towards 64-bit RISC OS

What driven the changes?

- Testing with a broad selection of applications found bugs.
- Porting old games made improvements to the system.
- Trying out different ways of doing things suggested better debug and implementations.
- Experimenting with architectures produced 64-bit RISC OS.

Towards 64-bit RISC OS

What driven the changes?

I've been asking people to try out the system, to iron out the bugs, but there has been very little interest. That has meant that I've had to do things myself. I've ported many random programs, just to see whether the system works. I've written software and tests to check the behaviour.
When a bug is found, I've introduced more tests to ensure that the behaviour does not regress in the future, and so that it's known about. You may remember that the Hexen and Heretic games were released a couple of years ago.
That was largely a challenge to check that the system was working sufficiently to run real games. As part of CI, the game is started and allowed to run for 10 seconds through RISC OS Pyromaniac to provide some smoke tests that show it's actually working.
My GitHub account has quite a few things on it which have exercised the system and become useful. More are still live on my own systems, as they're not really useful to anyone else.
And of course, I demonstrated a lot of the changes in RISC OS Pyromaniac in the 64-bit presentation last year, and of course this presentation is running in RISC OS 64.
But let's move on to those influences that drove the development...

Towards 64-bit RISC OS

6502 emulation

- I wanted to try out a 6502 emulator - running a system like the BBC Micro.
- Updated an existing open source emulator to use Unicorn-like interfaces.
- Restructured interfaces so that trapping system calls was easier.

I wanted to make it possible to run 6502 emulation inside RISC OS Pyromaniac.

- If you can run 6502 with RISC OS Pyromaniac, anything is possible.
- It's an intermediate step towards AArch64 or x64.

But actually it's a bit too alien to make work.

Towards 64-bit RISC OS

6502 emulation

We'll spin back to November 2023. I found a 6502 emulator on GitHub, and I started converting it over to use a Unicorn-like interface. The idea was to make it a little bit easier to work with, and to have interfaces in a similar form so that I could use it like Unicorn.
The original author had got it to the point that it could run BASIC, but I expanded on that and hooked more of the system so that it could interact more with the host. The reason for doing this was because I had this idea that I wanted to prove that RISCOS Pyromaniac could be used for other execution environments than ARM.
And 6502 is quite, quite alien. It could be quite interesting having a 6502 processor executing your code, but all of the operating system parts of it were actually implemented in RISC OS Pyromaniac.
That was the idea anyway. However, it was a little *too* alien at the time. Registers have different widths. Memory is limited to 64K. Stack is tiny, at just 256 bytes.
Whilst I had thought 'if I can make 6502 work, I can do anything', it turned out that it was harder than I had the energy for, with a full time job as well. As an intermediate step to x64 or AArch64, it didn't work.
I mentioned x64 there. I had the beginnings of a RISC OS x64 demonstration application which shows the rotating cog rendered by Draw. It kinda worked, but I wasn't confident enough with that work to do anything more with it. That was way back in April 2021, and I set it aside because the system wasn't really ready for such changes.

Towards 64-bit RISC OS

Replacing ARM with AArch64

- Update the disassembler system to support Thumb.
- Then update to support AArch64.
- Introduce DebuggerPlus features - widely used, and we don't want to clash.
- Debugger also supports the exception dump.
- Making the exception dump descriptive allows for other architectures.

Towards 64-bit RISC OS

Replacing ARM with AArch64

In December 2023, I began reworking the innards of RISC OS Pyromaniac to allow the emulation to be something other than ARM code. Whilst some parts of the system were flexible, and easy to replace their implementation, the emulator was rather core to the system and wasn't as easy to make switchable. The work is still ongoing, but it began to take shape as a system that could use a different architecture.

To help with this, the disassembler was extended. Previously, we could only disassemble ARM code. Obviously in the future, the emulator's tracing would need to be able to disassemble other architectures. The system was updated to allow Thumb to be disassembled. This allowed the SWI `Debugger_DisassembleThumb` to be implemented.

Shortly afterwards, the AArch64 disassembly was added, and a new SWI call `Debugger_DisassembleArch` was added to allow arbitrary architectures to be disassembled. At the same time, the Debugger module was also expanded so that it was aware of, and partially supported, the DebuggerPlus SWIs.

DebuggerPlus was created by Darren Salt, to change the form that the debugger's disassembly produced. Since those SWIs have been defined and used for many years, and I didn't want to clash, I skipped those SWIs. And then I implemented some of them, because... well, why not implement useful things?

Towards 64-bit RISC OS

Describing CPU registers

`OS_PlatformFeatures 64` describes the exception dump layout, to address the general problem for the future. The dump layout describes the architecture identifier, the length of instructions and the size of the dump block itself. And then for each register, we describe:

- The register's name.
- How offset the data is stored at.
- How wide the register is in bits.
- The alignment (eg the stack in AArch64 must be aligned to 4 bits).

For flags, we also describe where the flag is in a register, and its meaning.

Towards 64-bit RISC OS

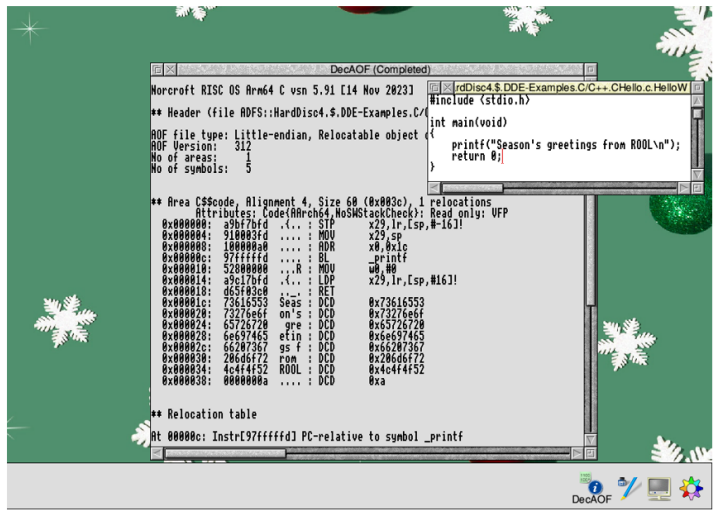
Describing CPU registers

Whilst updating the system to support AArch64 it became obvious that the exception registers were going to need an overhaul. The exception registers contain the details from any exception or other event which indicates the machine state at the time of failure.
In the 26-bit-world it had been defined to be 16 words long, one for each register. This was extended to 17 when the 32-bit extensions were created - adding the processor state register's value. For AArch64, it needed to be very different, as registers were a different width, had different names, and could be laid out differently.
I do not like fixing the immediate problem in front of me. If you fix just the issue in front of you, you only address a single instance of a problem. I like to address a class of problems, instead. That means that instead of dealing with just what's in front of me, I try to think about other cases that might be needed in the future. Things that other people might need. This allows much more flexibility for other uses.
Thinking about problems like that does mean that it's easy to over-engineer some things. However, as an operating system architect, I believe it's vital to provide functionality that others can use in ways you've never considered. By addressing the class of problem, you will always have a more generally useful system.
So, when it came to the exception dump area, I had to make it more future proof for other architectures. Instead of just saying "This is how it will be laid out for AArch64", I designed a new `OS_PlatformFeatures` interface that describes what the layout is. By describing the interface this way, it meant that if I got the layout wrong, I could just change the definition. And if someone decided to implement a 6502, or x64 or RISC-V system, they would merely need to change the layout definition.
It's only one part of the system, but it removes this from the problem scope in the future.

17 / 51

Towards 64-bit RISC OS

Christmas calendars 2023



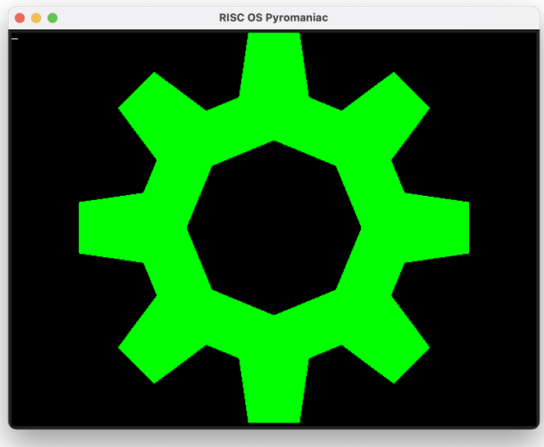
Towards 64-bit RISC OS

Christmas calendars 2023

Around this time, Iconbar ran a series of 'advent calendar' screenshots. ROOL submitted a screenshot to show AArch64 code being disassembled by DecAOF. I'm relatively sure that was a fake, as there would be no point in extending the Norcroft toolchain to support AArch64, and certainly not in AOF. But I had a system that was starting to be able to run programs, and could easily disassemble AArch64 to the same level. That felt pretty good.
Anyhow, updating an operating system to function with a different architecture isn't easy, so it took a while to feel comfortable with things working.

Towards 64-bit RISC OS

The first AArch64 RISC OS program



Towards 64-bit RISC OS

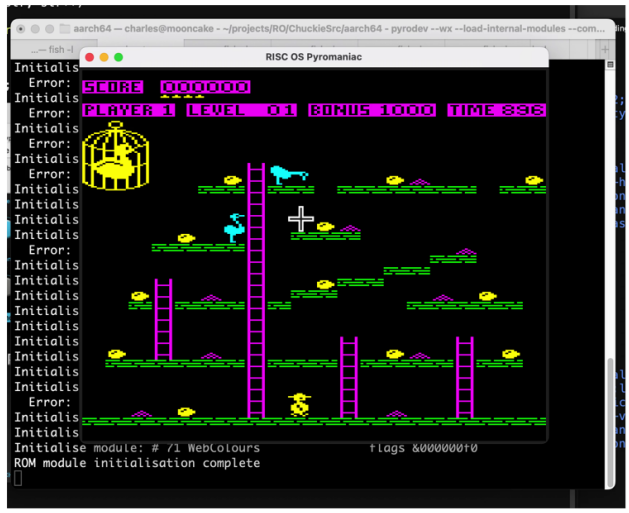
The first AArch64 RISC OS program

It was slow progress, but in June 2024, I finally got the first programs running in AArch64. A small C program that drew a cog using the Draw module and animated it rotating. I intended everything I was doing to be written in C - not using assembler unless absolutely necessary. To get this working, I'd been using the GCC compiler to compile my programs. There was a little bit of hackery and some dumb linker scripts, but an executable AIF file was running.

19 / 51

Towards 64-bit RISC OS

The first AArch64 RISC OS game



Towards 64-bit RISC OS

The first AArch64 RISC OS game

In July I got the first game running in RISC OS using AArch64 - Chuckie Egg. Michael Foot's conversion of the game to C was relatively easy to make work on RISC OS Pyromaniac. It amuses me that the first game that ran on RISC OS 64 was actually a game from the BBC Micro.

20 / 51

Towards 64-bit RISC OS

Presenting 64-bit RISC OS

- Those programs were made open source, on GitHub.
- A C library for 64-bit RISC OS was created on GitHub.
- I gave a presention on what I had done and decisions I had made at ROUGOL in August.
- I encouraged people that this was something we could do.

Towards 64-bit RISC OS

Presenting 64-bit RISC OS

Taking the work from these experiments, I created a new repository - riscos64-simple-applications - in GitHub in which to show off how things work. You'll find the example cog program there. This started to expand into a C library that can be used to run 64-bit RISC OS applications. Having got a C library, it was relatively easy to make my presentation tool work in RISC OS 64.

This was vital to my plans - there was a presentation to ROUGOL in August about 64-bit work, and I have a very strong need to demonstrate something that is a surprise. Finishing the presentation with the fact that the whole presentation was made using RISC OS 64 was key to that. It was very painful to get the ImageFileRender module working - I hadn't added AArch64 support to the module header generator at that time, so the interface was in hand-written assembler. It all came together in time though.

I had hoped that the presentation would spur people to talk about 64-bit and what could be done. It did generate some discussion, but most of the responses boiled down to either 'I'll believe it when it happens', 'Let's see what ROOL do', or 'there's no point if you're not going to support 32-bit applications'. It was a rather depressing result, to be honest. I can understand people being jaded, but after <mumble> years with nothing happening for 64-bit, I had expected more. Seeing that something was being done, and not only that but actual running programs, was - I thought - something that people should be excited about.

The 64-bit presentation also included links to all the resources I'd produced to date - the built 64-bit examples, and programs, including a port of Richard Russell's BBC BASIC, the C library and descriptions of all the decisions and technical proposals.

I always do this for my presentations. I try to ensure that I've got sufficient supporting information so that nobody can come back and say that I hadn't done enough. Or that I didn't provide what was needed to come back for more. This presentation is no exception.

Presentation resources

Presentation resources

If we pop off to my presentation site - talk.riscos.online - and then go to this presentation... you'll see there's the slides for the presentation, the full text of the presentation, and a whole load of links to things that I've talked about or are related to this presentation.
I'm not sure that people care about how much goes in to ensuring that what's being done is distributed, but it matters to me. Each of the presentations has these resources, and the Pyromaniac site includes even more in the form of videos and images through development.
And of course, there are the GitHub repositories where most of the publicly distributed things appear. I believe the more you make available to people, the better the project becomes. There is only so much you can do when you've got a full time job and family, but communication is key to the success of open source projects.

22 / 51

Towards 64-bit RISC OS

Creating a build environment

- All the 32-bit tools and libraries were already built into a Docker image for easier use.
- Extending this to 64-bit tools made the system much more useful.
- Took many, many iterations!

Towards 64-bit RISC OS

Creating a build environment

Anyhow, to build all these bits, I had created a Docker image for working with RISC OS builds, as this makes for a better distribution mechanism. The Docker image had most of the 32-bit tools that I use, and so I started creating a variant that included the 64-bit tools as well.
I've offered this around to people, to see if I could get any requests from people wanting to test out the system. Nobody was biting though. The build environment has continued and is more featureful than ever.

23 / 51

Towards 64-bit RISC OS

Live coding (1)

- Coding on YouTube live streams.
- Partly to improve my skills.
- Showing people how they can do things.
- To give the community more resources.
- Suggested ages ago by someone in a meeting.
- And because Matt Godbolt did it!

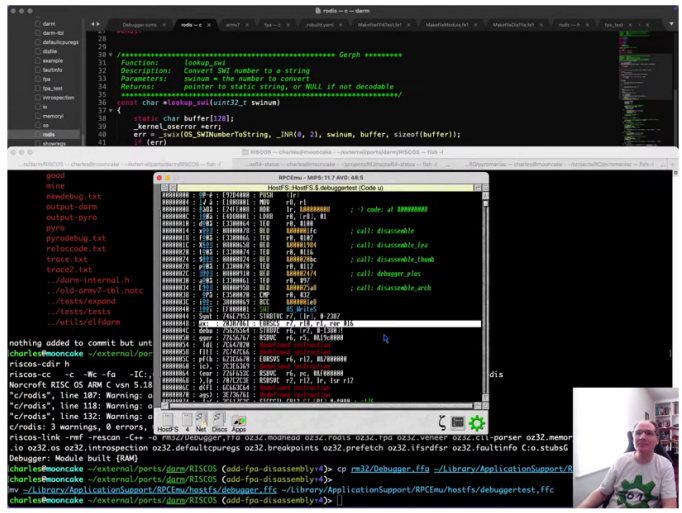
Towards 64-bit RISC OS

Live coding (1)

In parallel with this, I started doing some live coding - sharing my screen on YouTube for a few hours each week whilst I work on a project. I had decided to create a Debugger module which supported the ARM and AArch64 disassembly, similar to how I'd done with RISC OS Pyromaniac.
Why? Well. I enjoyed watching Matt Godbolt do live coding on this compiler explorer. It was really fascinating watching an expert do things live. I've never felt great about my presenting skills because I don't feel as comfortable in front of a camera. And I wanted to show that creating useful things was possible.
Plus someone, at a meeting a few years back had said that it would be better doing some things as videos. All these things came together when I decided I wanted to show how you created a RISC OS component - the debugger module.
Although that stalled after the 64-bit presentation, I renewed the sessions in 2025, and we now have a reasonable live following, and an increasing number of people who watch offline. I get questions about what is being done, and people point out mistakes. I really hope that seeing these things will encourage others that there were actual things happening, and that they might want to be involved.
I'll give a quick shout out here to Dave Thomas for his suggestions, and my series titles!
These sessions have now produced the Debugger module, the SystemVars module, and I'm currently working on the implementation of templates for the Window Manager.

Towards 64-bit RISC OS

Live coding (2)



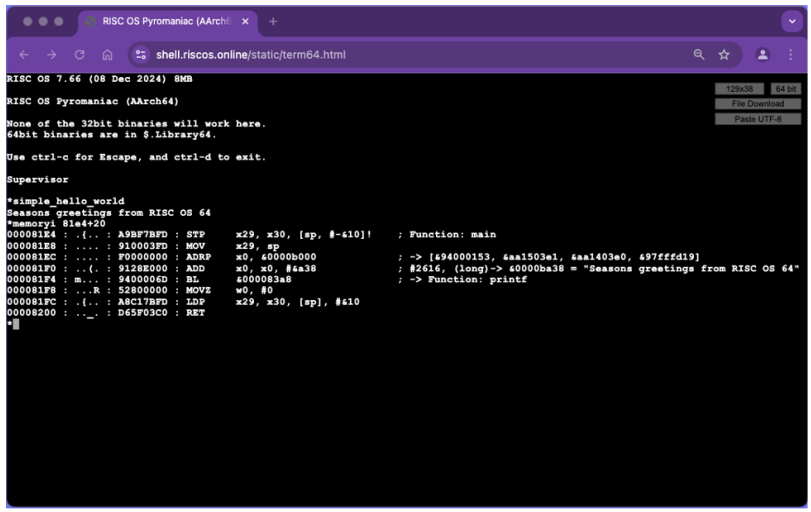
Towards 64-bit RISC OS

Live coding (2)

The Debugger was quite a long project, but it worked really well - I've got a version of the module which works well enough to use in !Zap. That's awesome. This is a screen grab from one of the sessions where I demonstrate the new module working in RISC OS within !Zap.
And doing this also found bugs in the RISC OS Classic Debugger module too, which was helpful.

Towards 64-bit RISC OS

Christmas calendars 2024



Towards 64-bit RISC OS

Christmas calendars 2024

In December 2024, Iconbar ran their 'advent calendar' screenshots series again.
This year, I submitted quite a few little examples. A christmasy snowman running in BASIC under RISC OS 64, a compilation and disassembly of the same code ROOL had done the previous year, an example of the online RISC OS shell system switching to 64-bit, and a 64-bit port of the original computer game - Adventure.
Here you can see the cloud based RISC OS shell running the 64-bit version of RISC OS, running and disassembling the C code that ROOL had posted.

Towards 64-bit RISC OS

Moonshots

> *"RISC OS is at risk of being left behind unless we act decisively."*

-- Thus spake ROOL, as they asked for £2.5 million.

Towards 64-bit RISC OS

Moonshots

RISC OS Open announced in early 2025 that they were opening a different class of bounty for work towards making components of RISC OS able to be ported to 64-bit systems.

...

27 / 51

Towards 64-bit RISC OS

Moonshots

> *"RISC OS is at risk of being left behind unless we act decisively."*

-- Thus spake ROOL, as they asked for £2.5 million.

- The Moonshots are intended to be larger blocks of work building towards a version of RISC OS that can be moved to 64-bit.
- They were looking for large donators to help with this.

I had done a lot of the basis for developing a 64-bit system, so I made a claim on their bounty.

...

At the time I was quite hopeful that this would mean that RISC OS Open would begin to work with the community, encouraging development, discussing plans and coordinating work to advance the operating system on modern systems. I was hopeful that, having discussed the issues, presented possible solutions, and demonstrated a functional version of RISC OS running 64-bit code, that they might wish to discuss what could be done.

28 / 51

Towards 64-bit RISC OS

My bounty claim (1)

Why ?

- Partly to raise awareness that I've been doing work towards this for years.
- Partly that if there's some money available, it'd be nice.
- Partly to highlight the amount of benefit it would be to work with others.

And of course, there was the fact that this would be done openly so that the community would see the co-operation.

Towards 64-bit RISC OS

My bounty claim (1)

When nothing came of that, I decided to push the issue by making a claim for a tiny portion of the work that I'd already done. As I'd been largely producing the system to directly address the issue of experimenting and moving RISC OS around, this seemed to be rather pertinent. If they weren't interested in coming to me, given my background, I would take what I had got to them.

29 / 51

Towards 64-bit RISC OS

My bounty claim (2)

What was it?

- Software and tools:
  - A 64-bit OS for testing and development.
  - Automation for modern CI development.
  - Tooling to build for 64-bit.
  - Documentation tailored towards documenting variants of the OS.
  - Test suites that exercise the current OS.
- Rationale:
  - Details of how this fits into the development process.
  - Evidence of what's been claimed.
- A claimed value.

Towards 64-bit RISC OS

My bounty claim (2)

I spent some time putting together a document describing the claim - what has been produced, how it fits into the goals of moonshots, what technical facilities I've provided, and a value on that. All backed up with documents, videos, articles, presentations and open source code to show how this fits together.
ROOL and I had a meeting to discuss this, and I answered their questions about the claim. They stated explicitly that the bounties would not be paid out for things that had already been done. However, they were interested in exploring possibilities and, despite seemingly not having understood how RISC OS Pyromaniac was useful, left with a better understanding. And I, for my part, made offers of the system, build environment and other tools.
It did seem quite positive, and I continued my work towards 64-bit, creating C versions of modules, updating Pyromaniac to 64-bit and doing the live coding to demonstrate how to create modules that ran on 32-bit and 64-bit, replacing the regular RISC OS modules.
At the same time I made sure that I was communicating what I was doing to the community through the ROOL forums (and obviously by publishing to GitHub as well). This is, I think, the primary place where RISC OS users will be finding out what's being developed, and the best place to engage when things are happening.
But nothing was forthcoming from RISC OS Open about progressing RISC OS, or in producing any plan for how to get there.

30 / 51

3. Changing my approach

3. Changing my approach

If things aren't working, try a different way...

31 / 51

Changing my approach

What's going wrong? (1)

- Things hadn't worked out with ROOL (still waiting for them to provide guidance, or... anything really).
- The community hadn't moved to start towards 64-bit.
- My technical things aren't making any impact...

So let's work out how to change the approach...

Changing my approach

What's going wrong? (1)

Having made very little progress on involving people, and with ROOL not being involved, I decided to change my approach.
ROOL weren't interested in what I had to offer, and weren't going to offer any guidance on what they wanted to happen. Nobody else was working towards 64-bit, and my technical things weren't making an impact.

32 / 51

Changing my approach

What's going wrong? (2)

I can't change ROOL, but why is the community not on board?

- Apathy.
- Lack of direction.
- Waiting for something to happen. Or just for ROOL.
- Scepticism.
- Resignation to their fate.
- No need to change.
- Just wanting to use the system.
- Longing for things from 30 years ago.
- Lack of time and inclination.

Changing my approach

What's going wrong? (2)

I decided to look at what was missing, and why I believe the community wasn't engaged...

- There is a degree of apathy.
- There is a lack of focus.
- Some people are waiting for something to happen.
- There is scepticism that anything will happen.
- There are those that feel resigned that it's a lost cause.
- There are those that don't see a need to change.
- There are those who just want to use the system.
- There are some who just wish that things were as they were 30 years ago.
- There is often a lack of time and inclination - many of us have full time jobs and families.

Mix and match these in a small community, and I think you have a pretty good definition of a RISC OS user's attitude.
I can't change people, but I can affect their perception of things. As I've said, I want to build things that people use. But to do that, you need to make sure that they know about those things. You need to engage with people when they are involved, and you need to ensure that they have the ability to be able to do things.

Changing my approach

What's going wrong? (3)

What have I been doing?

- Communicating.
- Open sourcing software.
- Demonstrating.
- Presenting and live coding.
- Providing encouragement.

Changing my approach

What's going wrong? (3)

What I had been doing was in line with this:

- Communicating what has been done. Reasonably regular announcements of things, website updates with details of how to use tools.
- Making software available. I've open sourced many tools and programs, to allow others to make use of them. Placing them on GitHub seemed the best way to do this. It makes sure that they're available and people can get involved if they want.
- Demonstrating things. Providing videos, screenshots, and descriptions of software and plans, so that people can see what it means. Not everyone takes things in in the same way, so offering different media can be helpful.
- Doing presentations, and live coding. Hopefully this provides a demonstration of how to do things. And of the ways in which things go wrong - it's useful to show people that mistakes aren't terrible, and even if things go wrong sometimes it turns out to be helpful.
- Providing encouragement. All of these ways show that things are moving, and that development isn't stagnant.

This addresses some of the areas, but it hasn't been enough to reach people. Sometimes even if people see what you're selling, they're not going to be interested. But for the rest, maybe there were areas that I could still improve.
The live coding had been intended to improve my skills in presenting live, and to encourage people to try things, even if they're not their area. There were still some things I could do to try to encourage people who were waiting for something to happen, who were sceptical, or who didn't see any focus.
To try to address this, I expanded my work to include setting out a clearer plan, and to provide regular updates. Maybe this would hit some of the things that are missing in the community, and provide enough encouragement for people to get involved.

Changing my approach

Make a plan. Execute it. (1)

- There's no written plan for RISC OS Pyromaniac.
- There's only me working on it.
- Having a plan for RISC OS 64 work would hit some of the issues I just mentioned.

Of course, just having a plan doesn't magically make people get on board... but it contributes to the viability of the project.

Changing my approach

Make a plan. Execute it. (1)

Creating RISC OS Pyromaniac had been easy because *I* knew what I needed to do. However, the work towards RISC OS 64 was different. I couldn't do it all on my own, and I didn't see any need to. But I did have a reasonable idea of what was needed and how it might fit together.

35 / 51

Changing my approach

Make a plan. Execute it. (2)

A plan will...

- Define what needs to be done.
- Allow problems to be ironed out early.
- Show that things have been thought about.
- Identify deficiencies.
- Provide encouragement to help out.
- Offers guidance for what's needed.
- Give confidence that progress can be made.

... and avoids the current situation where people just hope that something will magically appear.

Changing my approach

Make a plan. Execute it. (2)

Having a plan can help you to work out how difficult a project is. It can iron out problems before you hit them. It can help others to see how to do things. It can allow you to do things at the right time - not 'putting the cart before the horse', as the saying goes. And it can give you confidence that things are moving - that there is progress and how far you've come.

Creating a plan would meet, or at least contribute to, some of the parts that I saw as problems. And a plan would allow me to see that I was achieving something.

So I started working out a skeleton of a plan. What I've produced isn't a project plan - it doesn't set out everything, and doesn't break down everything or show all the issues that will be encountered. If I were to do that, I'd have one of the best managed RISC OS projects... by about 2030. Instead I've tried to work out how things might be developed, and what phases these could fall into.

The goal of *producing* the plan (not the goal of the plan itself - maybe that's a little meta, but hear me out!) was to...

- Show that there is a plan - there *is* some focus.
- Show progress - encouraging people that there is something happening.
- Break down some recognisable milestones - giving something to look forward to.
- Provide encouragement - maybe people can get involved in something large.
- Explain how to progress - guidance for anyone who wants to get involved.
- Limit the depth - only provide enough to meet the needs, and we can get into the weeds as we approach each part.

That meets some of the things that are missing from the RISC OS 64 work.

And I think that's gone quite well so far. The RISC OS 64 Status Wiki now describes the work needed. I've broken things down into phases. Each phase has something of a focus, and goals, but tries to show progress in other areas. I've defined some terms so that I can talk consistently about the development process. And for some areas, we've got more information about what is involved.

Changing my approach

Work out what we mean

There are different types of components...

- Original assembler components - which need a complete re-write.
- Libraries and prototypes - a prototype that shows how to build a filesystem is more valuable to developers.
- Original C components - which will need updating to be aware of 64-bit.

Each of these types of components will have a different development path, but we can still track them.

Changing my approach

Work out what we mean

What do I mean by this?
I've said I've defined some terms, so let's give them some meaning. During the development of components and features, it's important to see how you're progressing - and not every component or feature will go the same way.

- Original assembler modules. For some components, you'll design it, build some basic functionality, wire it up to an application or module and then test it.
- Prototypes and libraries

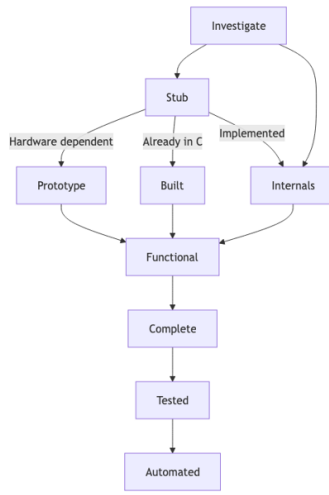
For others, you only want to get as far as some prototype code, so that you can give it to other developers to do something with. I'm thinking of things like hardware stubs, or common libraries. Making the RISC OS part common means that other people can handle the specific implementations. Consider something like the Filesystem interfaces - making a stub to work with them means that anyone can write a filesystem by just plugging their bits in.

- Original C components

And then there's another path for the components that are already implemented, but need to be made to work in a 64-bit environment.

Changing my approach

Software Development Lifecycle



Changing my approach

Software Development Lifecycle

Here's a diagram that shows the way that each component can be developed, and the stages they go through.

- Investigate - decide how to proceed
- Stub - just the interface to the OS; no implementation.
- Prototype - largely functional, but hardware implementation missing.
- Built - component has been built into a binary, but it's not understood whether it is working, or even useful.
- Internals - internal implementation, but no OS wiring.
- Functional - wired from OS interface to internals, but may be missing less used features, including I18N.
- Complete - implemented completely.
- Tested - implemented and tested manually.
- Automated - testing has been automated.

The idea is to track the development process so that we can quantify how much has been done, and to be able to see what needs doing when.
I'm focussing initially on the components of the ROM. I have broken down the components for the ROM into a number of phases as I mentioned.

RISC OS 64 status tour

RISC OS 64 status tour

Let's have a quick delve into the status pages, and I'll give a brief tour.
The main index has links to all the details that have been brought together here. Each phase has a nice percentage, to give a quick overview of where we're up to. As you can see, there's actually a reasonable amount done on phase 1.
So, let's have a quick look at what Phase 1 is made up of...
The definition of the phase is given as a summary on the page. Basically, phase 1 is intended to show people that work is happening and to bring up some of the core modules so that they can be used. You'll remember I mentioned that the question was asked about the version of RISC OS that was being implemented?
Well, I'm choosing to implement the style of modules that were in RISC OS Select - breaking out parts of the kernel into separate components. Why? Because I've already spent the time working out a separation that allows independent components to be implemented without having to build a monolithic kernel.
So we have the Kernel components for system variables, conversions, commands, evaluation and some other parts as separate modules. In the live coding, I implemented the bulk of the SystemVars module. The table here shows how far I expect each component to get within phase 1.
As you can see there are quite a few components. If you've been looking at this before you might be wondering why components like FileSwitch aren't high up on the list in phase 1?
The answer's pretty simple. They don't need to be. We already have a working FileSwitch in 32-bit RISC OS, and RISC OS Pyromaniac supports file access in both 32-bit and 64-bit versions. So we don't need to worry about that right now. By moving other components on, and not being bogged down by that very complex component, we can show progress and get some more varied components working first.
That's one of the advantages of having RISC OS Pyromaniac - we can test out components using the environment even though we don't have the whole system present. It's kinda like how Acorn had the ARM board running as the tube whilst the BBC provided the I/O system.

Changing my approach

Communicating the status

- The repository is updated regularly.
- Components and phases have more documentation added as I find I need it.
- Status is visible to all.
- Anyone wanting to comment can open issues, or ask questions.

If it's not sufficient, more detail can always be added.

Changing my approach

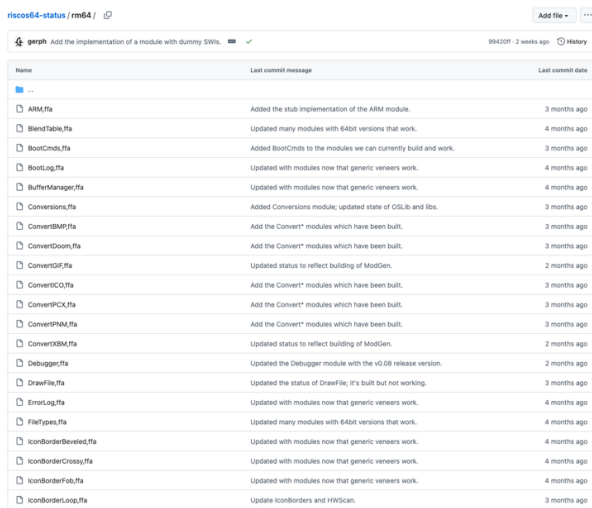
Communicating the status

I'll not go through the other components and phases, but just say that further into the future, the less firm the plans are, and less detail present. Some things might need moving, and many will need refining with more information.
The idea is that hopefully the later phases will have more people than just myself doing the work. I'm hoping that the encouragement will get some people interested in getting involved in working on developing things. Maybe they won't, and if they don't, then at least I'll have tried to do something.
As I've been developing the components - either on my own or as part of the live coding - I've tried to include tests. I really want to be able to show that 64-bit components are working, and avoid issues. Or at least to know where they lie. This status repository is no different in that respect. I've included built versions of the modules and absolutes within the repository, so that people can try them out.

40 / 51

Changing my approach

Testing new components (1)



riscos64-status / rm64 /

Add file

gerph Add the implementation of a module with dummy SWIs. 8942Off · 2 weeks ago History

Name	Last commit message	Last commit date
..		
ARM,ffa	Added the stub implementation of the ARM module.	3 months ago
BlendTable,ffa	Updated many modules with 64bit versions that work.	4 months ago
BootCmds,ffa	Added BootCmds to the modules we can currently build and work.	3 months ago
BootLog,ffa	Updated with modules now that generic veneers work.	4 months ago
BufferManager,ffa	Updated with modules now that generic veneers work.	4 months ago
Conversions,ffa	Added Conversions module; updated state of OSLib and libs.	3 months ago
ConvertBMP,ffa	Add the Convert* modules which have been built.	3 months ago
ConvertDoom,ffa	Add the Convert* modules which have been built.	3 months ago
ConvertGIF,ffa	Updated status to reflect building of ModGen.	2 months ago
ConvertICO,ffa	Add the Convert* modules which have been built.	3 months ago
ConvertPCX,ffa	Add the Convert* modules which have been built.	3 months ago
ConvertPNM,ffa	Add the Convert* modules which have been built.	3 months ago
ConvertXBM,ffa	Updated status to reflect building of ModGen.	2 months ago
Debugger,ffa	Updated the Debugger module with the v0.08 release version.	2 months ago
DrawFile,ffa	Updated the status of DrawFile; it's built but not working.	3 months ago
ErrorLog,ffa	Updated with modules now that generic veneers work.	4 months ago
FileTypes,ffa	Updated many modules with 64bit versions that work.	4 months ago
IconBorderBeveled,ffa	Updated with modules now that generic veneers work.	4 months ago
IconBorderCrossy,ffa	Updated with modules now that generic veneers work.	4 months ago
IconBorderFob,ffa	Updated with modules now that generic veneers work.	4 months ago
IconBorderLoop,ffa	Update IconBorders and HWScan.	3 months ago

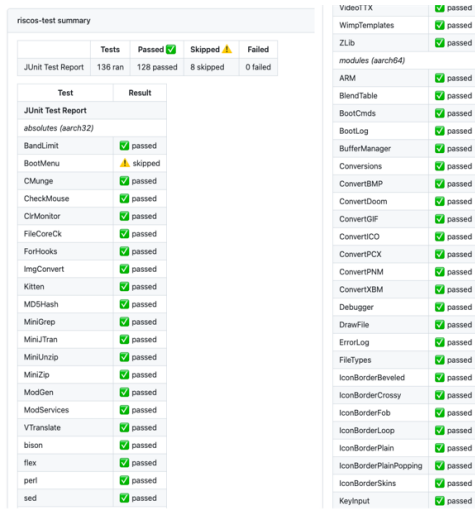
Changing my approach

Testing new components (1)

If we look in `rm64`, we see there are a number of built modules which run in RISC OS 64. How do we know that they run?

Changing my approach

Testing new components (2)



riscos-test summary

	Tests	Passed ✅	Skipped ⚠️	Failed
JUnit Test Report	136 ran	128 passed	8 skipped	0 failed

Test	Result
JUnit Test Report	
absolutes (aarch32)	
BandLimit	✅ passed
BootMenu	⚠️ skipped
CMunge	✅ passed
CheckMouse	✅ passed
ClrMonitor	✅ passed
FileCoreCk	✅ passed
ForHooks	✅ passed
ImgConvert	✅ passed
Kitten	✅ passed
MD5Hash	✅ passed
MiniGrep	✅ passed
MiniJTran	✅ passed
MiniUnzip	✅ passed
MiniZip	✅ passed
ModGen	✅ passed
ModServices	✅ passed
VTranslate	✅ passed
bison	✅ passed
flex	✅ passed
perl	✅ passed
sed	✅ passed
VideoTTX	✅ passed
WimpTemplates	✅ passed
ZLib	✅ passed
modules (aarch64)	
ARM	✅ passed
BlendTable	✅ passed
BootCmds	✅ passed
BootLog	✅ passed
BufferManager	✅ passed
Conversions	✅ passed
ConvertBMP	✅ passed
ConvertDoom	✅ passed
ConvertGIF	✅ passed
ConvertICO	✅ passed
ConvertPCX	✅ passed
ConvertPNM	✅ passed
ConvertXBM	✅ passed
Debugger	✅ passed
DrawFile	✅ passed
ErrorLog	✅ passed
FileTypes	✅ passed
IconBorderBeveled	✅ passed
IconBorderCrossy	✅ passed
IconBorderFob	✅ passed
IconBorderLoop	✅ passed
IconBorderPlain	✅ passed
IconBorderPlainPopping	✅ passed
IconBorderSkins	✅ passed
KeyInput	✅ passed

Changing my approach

Testing new components (2)

If we have a look at the GitHub Actions and locate the 'Test all binaries' job, we can see the results of loading these modules and absolute files. They're tested in both 32-bit and 64-bit RISC OS and report their status. It's a rudimentary test, but each component's repository will have a set of more comprehensive tests.

Changing my approach

Testing without an OS?

- We have an OS - we have two.
- We have RISC OS Classic - for legacy.
- We have RISC OS Pyromaniac - for 32-bit and 64-bit.
- We have the RISC OS Build service to test things.

This gives us a means to test manually and automatically relatively easily.

Changing my approach

Testing without an OS?

How do we test things without an OS to run them on?
That's easy - we *have* an OS to run them on. RISC OS Pyromaniac supports 32-bit and 64-bit and the RISC OS Build service can run both architectures. So we just fire off the module and a test script to the service, and we can get back whether it worked, or whether it crashed.
Because I've already implemented tests for a good amount of RISC OS as part of RISC OS Pyromaniac, these can be reused for the new components. In fact they are used to test the components that I've built so far. In some cases they haven't needed any changes at all.

43 / 51

Changing my approach

Developing for a new OS?

Developing for a new OS needs tools...

- Norcroft C compiler only targets ARM.
- Better compilers exist which have mass usage and development.
- GCC can be made to work for RISC OS.
- If we remove the RISC OS-specific needs, it's even easier.

The Docker image that I've created allows building of RISC OS components in both 32-bit and 64-bit very simply.

Changing my approach

Developing for a new OS?

I've said that we can build 64-bit RISC OS components, but haven't discussed how we build them. Obviously we've got the Norcroft toolchain to build 32-bit components, but there isn't a version of Norcroft that builds 64-bit code.
That's fine because other, far better, solutions exist. It would be insane to make Norcroft build 64-bit code when we can use GCC, CLang, TinyCC or any of the other lesser known systems to build C code. And that's before we consider other languages.
Wrangling GCC to work in a way that operates with RISC OS isn't too hard. Doing so with 64-bit code is a little more tricky. Together with creating a new C library that we can use for RISC OS 64, it's taken quite a few months, but I've built some pretty good tooling. I've been looking for anyone to test this (like the other many things I've built) but haven't had any takers. The build environment uses Docker to let me build software in a simple way. I've set it up so that building for 32-bit and 64-bit is just as simple.
When I say 'simple', I mean that you just give a different parameter to AMU and you get out a 64-bit binary.

Build environment demonstration

Build environment demonstration

Let me give a short demonstration.
Let's fire up a simple terminal, and we'll create a directory for this little project. I wrote a software project creation tool to make creating a program easy - `riscos-project`. We'll run that. And we'll create some skeleton code.

```
riscos-project create --name Test --type command --skeleton
```

We can see that we've created a C file and a Makefile. If we see what's in that file, we can see that it's a very simple program that just prints some messages. Let's build it.

```
riscos-amu
```

Ok it's built. How about we run that, just to prove it's working?

```
pyrodev --common --command aif32.test
```

Want to see what's in it?

```
less aif32/Test,ff8
```

Let's quickly search for the `main` function... there we are, and you can see that it's printing its messages.
Ok, but I said we can build this as 64-bit. So let's do that.

```
riscos-amu BUILD64=1
```

There we are, we have a 64-bit RISC OS binary. Again, let's run it.

```
pyrodev --64 --command 'aif64.test'
```

And again, let's see what's in it.

```
less aif64/Test,ff8
```

45 / 51

Changing my approach

Native languages

Cross-compiling is good, but we also want languages running in RISC OS...

- **SDL BASIC** - rudimentary port.
- **MyBASIC** - lightweight interpreter port.
- **SED** - it's a language, honest.
- **PocketPython** - Python for small systems.
- **Perl** - yeah, I like Perl.
- **HUProlog** - might as well have a functional language.
- **Lua** - rudimentary port, but surprisingly easy.
- **PicoC** - C interpreter. I've even added `_swix`.
- **Berry** - a small scripting language.
- **TCC** - C compiler, targetting AArch64.

Changing my approach

Native languages

Whilst cross-compiling is good, it's also important to have languages available for the system itself, so that users can do things. Whilst it's not something I'm focusing on, it's important to ensure that the community can see that it's not just a bare system which you have to bring your own tools to.
I've made a few languages work in RISC OS 64, to varying degrees. These are documented in the status repository, but a quick round up...

- SDL BASIC - I built Richard Russell's BASIC as one of the first ports to RISC OS 64. It's functional, though I've not spent much time with it.
- My BASIC - This is a lightweight BASIC interpreter. I've not done much with this either.
- SED - You might not think it, but SED is a Turing-complete language.
- PocketPython - I ported Pocket Python, really just to see how hard it would be. It only took a few hours.
- Perl - The old Perl 5 that I use works under RISC OS 64 as well.
- Prolog - I ported HUProlog from the old HENSA sources, largely just so that I had a functional language on the list.
- Lua - I had hunted for some small language processors, and then remembered the Lua is intended to be embeddable and portable. So ... that works too.
- PicoC - PicoC is a C interpreter, which you can use as a command line tool or with C files. I've added the `_swix` interface, so it's actually handy for testing modules in RISC OS 64.
- Berry - Berry is a small scripting language. Might be useful for doing simple coding in.
- TCC - a C compiler targeting AArch64.

And then there's the BBC BASIC implementation that Steve Revill is working on. He's not ready to offer any progress statements, or share it yet, but has been working on it. As we heard at the London show, he's not going to do any more unless he can get paid for it.

RISC OS 64 language demonstration

RISC OS 64 language demonstration

Shall we have a quick look at some of the languages?
Let's start with PocketPy.

```
cd ~/external/ports/pocketpy
cd riscos
pyrodev --64 --gos
appsize 20000k
aif64.pocketpy
print("Hello world")
```

Let's try Pico C...

```
cd ../picoc
pyrodev --64 --gos
dir riscos
aif64.picoc -i
printf("Hello world\n");
_swix(2, _IN(0), "Hello world\n");
```

And Lua?

```
cd ../lua/riscos
pyrodev --64 --gos
print("Hello world")
os.time()
```

Anyhow, you get the idea.
All these mean that there are a number of languages that will be available from day 1 to run on a 64-bit version of the OS.

4. Conclusion

4. Conclusion

Let's wrap this up, shall we?

48 / 51

Conclusion

I've talked about many how the RISC OS Pyromaniac project has evolved.
I've talked about what I'm trying to do to address the issues I see.

- Built up a plan.
- Communicated what's being done.
- Explaining the details.
- Videos to show development and build a community.
- Tools and testing environments for people to use.

And I'm doing this because I hope to encourage others.

Conclusion

I've talked quite a bit about RISC OS Pyromaniac and what this project has evolved into. As you should all know by now, my motivation is to push RISC OS forwards. To encourage the community and make resources as available as possible.
You've heard about what I've been doing. You've heard about where I see problems, and what I have done to try to address them.

- Offering a plan for how the 64-bit work can be progressed.
- Communicating what's happening with example code, output and videos.
- Explaining and documenting technical details.
- Providing videos to show development and generate interest in the 64-bit work.
- Providing tools and testing environments for people to use.

I hope this is encouraging to people, and that they will get involved in developing RISC OS for the future. I can only ask people to get involved, and provide the tooling and guidance.

49 / 51

Conclusion

Are *you* interested in being involved?

Conclusion

Are *you* interested in being involved in developing for the future of RISC OS?

Questions

I'll take any questions that people have.
Slides and Info: **https://presentation.riscos.online/pyromaniac4/**

Questions

Thank you very much, I'll take questions now...