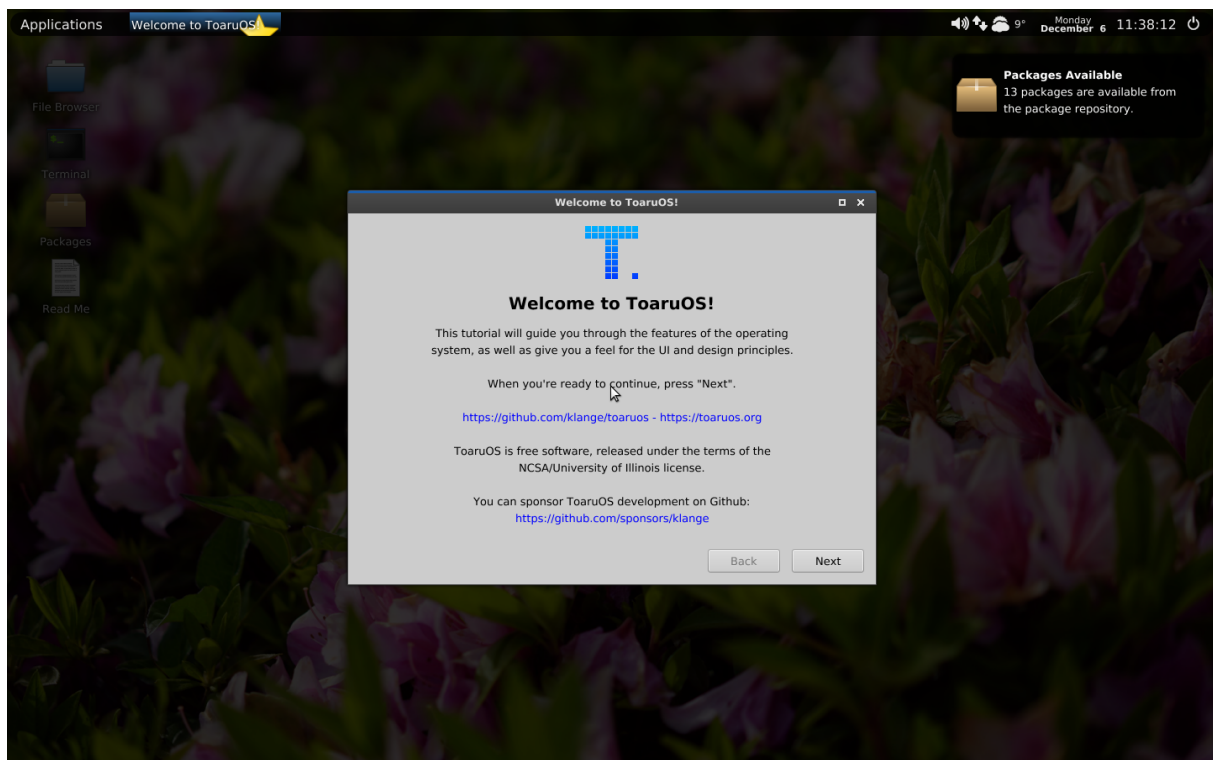

ToaruOS

ToaruOS is a “complete” operating system for x86-64 PCs and experimental support for ARMv8.

While many independent, hobby, and research OSes aim to experiment with new designs, ToaruOS is intended as an educational resource, providing a representative microcosm of functionality found in major desktop operating systems.

The OS includes a kernel, bootloader, dynamic shared object linker, C standard library, its own composited windowing system, a dynamic bytecode-compiled programming language, advanced code editor, and dozens of other utilities and example applications.

There are no external runtime dependencies and all required source code, totalling roughly 100k lines of (primarily) C, is included in this repository, save for Kuroko, which lives separately.



Demonstration of ToaruOS’s UI and some applications.

History

I have been working on ToaruOS for over a decade now, and my goals have changed over the years.

When I first started the project in December 2010, my aim was to “learn by doing” - studying Unix-

like systems by making one from scratch. I had been a contributor to Compiz, one of the first widely-used compositing window managers for X11, a few years prior, and somewhat naturally ToaruOS gained a GUI built on similar concepts early on.

For its original 1.0 release in 2015, ToaruOS was not the “completely from scratch” OS it has since become. Newlib provided the libc, and the GUI was built on Cairo, libpng, and Freetype. In the middle of 2018, I started a new project to replace these third-party components, which was eventually completed and merged to become ToaruOS 1.6.

Through out the project, ToaruOS has also attracted quite a few beginner OS developers who have tried to use it as a reference. ToaruOS’s kernel, however, was a source of personal embarrassment for me, and in April 2021, after a long hiatus, I began work on a new one. The result was Misaka: a new 64-bit, SMP-enabled kernel. Misaka was merged in May and started the 1.99 series of beta releases leading up to ToaruOS 2.0.

Features

- **Dynamically linked userspace** with shared libraries and [dlopen](#).
- **Composited graphical UI** with software acceleration and a late-2000s design inspiration.
- **VM integration** for absolute mouse and automatic display sizing in VirtualBox and VMware Workstation.
- **Unix-like terminal interface** including a feature-rich terminal emulator and several familiar utilities.
- **Optional third-party ports** including GCC 10.3, Binutils, SDL1.2, Quake, and more.

Notable Components

- **Misaka** (kernel), kernel/, a hybrid modular kernel, and the core of the operating system.
- **Yutani** (window compositor), apps/compositor.c, manages window buffers, layout, and input routing.
- **Bim** (text editor), apps/bim.c, is a Vim-inspired editor with syntax highlighting.
- **Terminal**, apps/terminal.c, xterm-esque terminal emulator with 24-bit color support.
- **ld.so** (dynamic linker/loader), linker/linker.c, loads dynamically-linked ELF binaries.
- **Esh** (shell), apps/esh.c, supports pipes, redirections, variables, etc.
- **Kuroko** (interpreter), kuroko/, a dynamic bytecode-compiled programming language.

Current Goals

The following projects are currently in progress:

-
- **Rewrite the network stack** for greater throughput, stability, and server support.
 - **Improve SMP performance** with better scheduling and smarter userspace synchronization functions.
 - **Support more hardware** with new device drivers for AHCI, USB, virtio devices, etc.
 - **Bring back ports** from ToaruOS “Legacy”, like muPDF and Mesa.
 - **Improve POSIX coverage** especially in regards to signals, synchronization primitives, as well as by providing more common utilities.
 - **Continue to improve the C library** which remains quite incomplete compared to Newlib and is a major source of issues with bringing back old ports.
 - **Replace third-party development tools** to get the OS to a state where it is self-hosting with just the addition of a C compiler.
 - **Implement a C compiler toolchain** in toarucc.

Building / Installation

Building With Docker

General users hoping to build ToaruOS from source are recommended to fork the repository on Github and make use of the Github CI pipeline.

For those looking to build locally on an appropriately configured Linux host with Docker, a build container is available. The ToaruOS repository should be used as a bind mount at `/root/misaka` and `util/build-in-docker.sh` can be run within this container to complete the compilation process:

```
1 git clone https://github.com/klange/toaruos
2 cd toaruos
3 git submodule update --init kuroko
4 docker pull toaruos/build-tools:1.99.x
5 docker run -v `pwd`: /root/misaka -w /root/misaka -e LANG=C.UTF-8 -t
  toaruos/build-tools:1.99.x util/build-in-docker.sh
```

After building like this, you can run the various utility targets (`make run`, etc.). Try `make shell` to run a ToaruOS shell using a serial port with QEMU.

Build Process Internals

The `Makefile` uses a Kuroko tool, `auto-dep.krk`, to generate additional Makefiles for the userspace applications and libraries, automatically resolving dependencies based on `#include` directives.

In an indeterminate order, the C library, kernel, userspace libraries and applications are built, combined into a compressed archive for use as a ramdisk, and then packaged into an ISO9660 filesystem image.

Project Layout

- **apps** - Userspace applications, all first-party.
- **base** - Ramdisk root filesystem staging directory. Includes C headers in `base/usr/include`, as well as graphical resources for the compositor and window decorator.
- **boot** - BIOS and EFI loader with interactive menus.
- **build** - Auxiliary build scripts for platform ports.
- **kernel** - The Misaka kernel.
- **kuroko** - Submodule checkout of the Kuroko interpreter.
- **lib** - Userspace libraries.
- **libc** - C standard library implementation.
- **linker** - Userspace dynamic linker/loader, implements shared library support.
- **modules** - Loadable driver modules for the kernel.
- **util** - Utility scripts, staging directory for the toolchain (binutils/gcc).
- **.make** - Generated Makefiles.

Filesystem Layout

The root filesystem is set up as follows:

- `bin`: First-party applications.
- `cdrom`: Mount point for the CD, if available.
- `dev`: Virtual device directory, generated by the kernel.
 - `net`: Network interface devices.
 - `pex`: Packet Exchange hub, lists accessible IPC services.
 - `pts`: PTY secondaries, endpoints for TTYs.
- `etc`: Configuration files, startup scripts.
- `home`: User directories.
- `lib`: First-party libraries
 - `kuroko`: Kuroko modules.
- `mod`: Loadable kernel modules.
- `proc`: Virtual files that present kernel state.

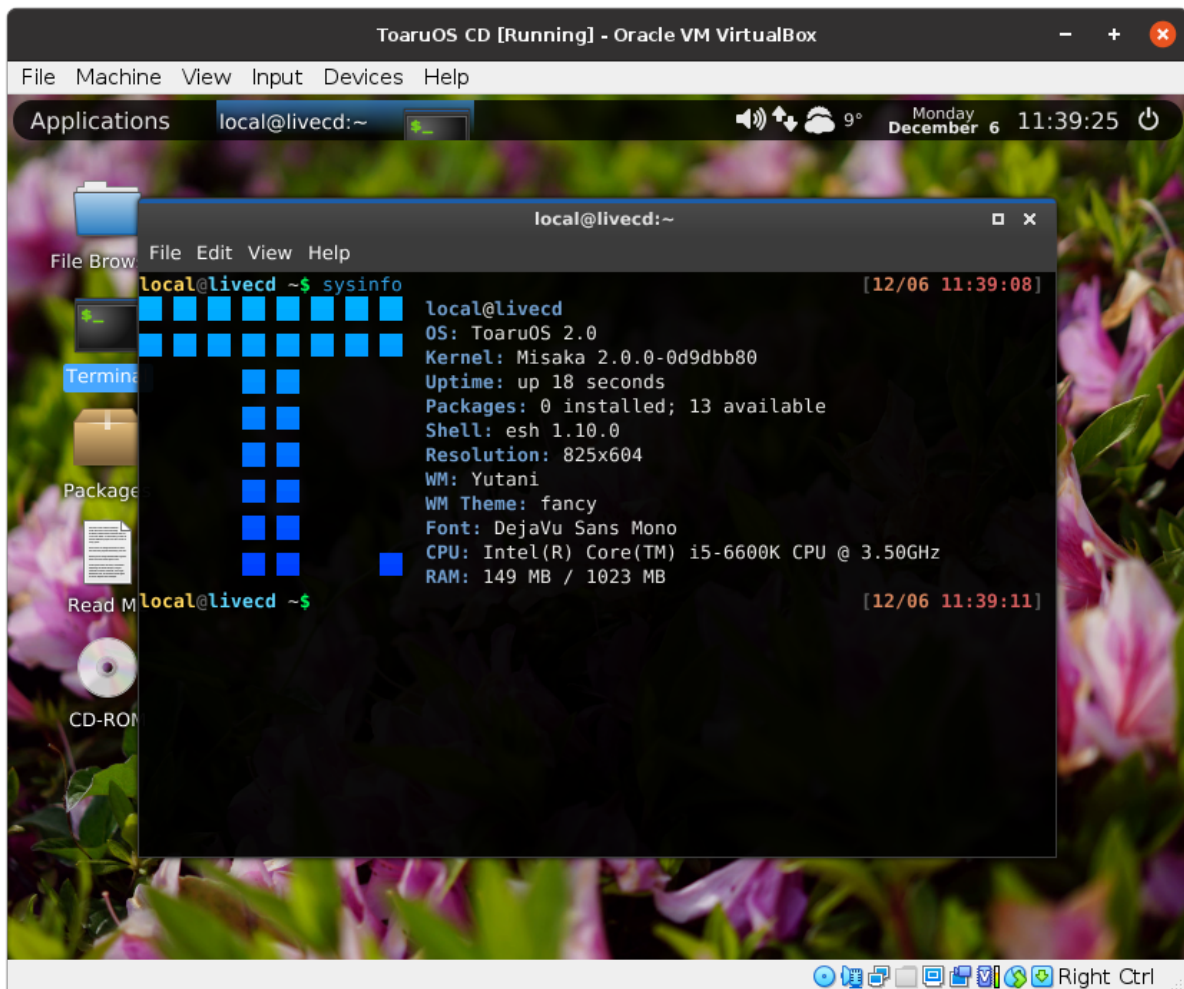
-
- 1, etc.: Virtual files with status information for individual processes.
 - `src`: Source files, see “Project Layout” section above.
 - `tmp`: Mounted as a read/write tmpfs normally.
 - `usr`: Userspace resources
 - `bin`: Third-party applications, normally empty until packages are installed.
 - `include`: Header files, including potentially ones from third-party packages.
 - `lib`: Third-party libraries. Should have `libgcc_s.so` by default.
 - `share`: Various resources.
 - * `bim`: Syntax highlighting and themes for the text editor.
 - * `cursor`: Mouse cursor sprites.
 - * `fonts`: TrueType font files. Live CDs ship with Deja Vu Sans.
 - * `games`: Dumping ground for game-related resource files, like Doom wads.
 - * `help`: Documentation files for the Help Browser application.
 - * `icons`: PNG icons, divided into further directories by size.
 - * `ttk`: Spritesheet resources for the window decorator and widget library.
 - * `wallpapers`: JPEG wallpapers.
 - `var`: Runtime files, including package manager manifest cache, PID files, some lock files, etc.

Running ToaruOS

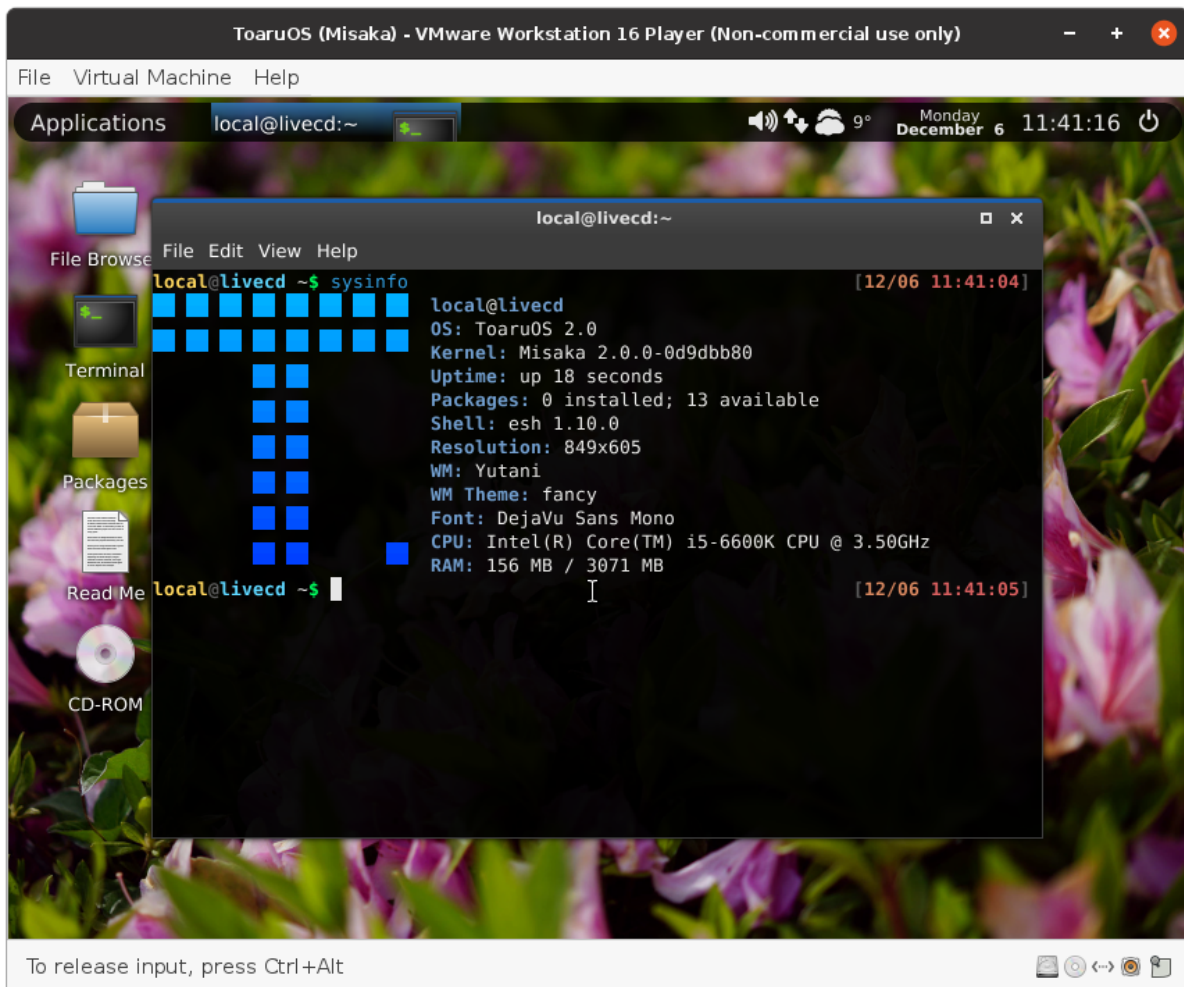
VirtualBox and VMware Workstation

The best end-user experience with ToaruOS will be had in either of these virtual machines, as ToaruOS has support for their automatic display sizing and absolute mouse positioning.

Set up a new VM for an “other” 64-bit guest, supply it with at least 1GiB of RAM, attach the CD image, remove or ignore any hard disks, and select an Intel Gigabit NIC. Two or more CPUs are recommended, as well.



ToaruOS running in VirtualBox.



ToaruOS running in VMware Workstation Player.

By default, the bootloader will pass a flag to the VirtualBox device driver to disable “Seamless” support as the implementation has a performance overhead. To enable Seamless mode, use the bootloader menu to check the “VirtualBox Seamless” option before booting. The menu also has options to disable automatic guest display sizing if you experience issues with this feature.

QEMU

Most development of ToaruOS happens in QEMU, as it provides the most flexibility in hardware and the best debugging experience. A recommended QEMU command line in an Ubuntu 20.04 host is:

```
1 qemu-system-x86_64 -enable-kvm -m 1G -device AC97 -cdrom image.iso -smp
2
```

Replace `-enable-kvm` with `-accel hvm` or `-accel haxm` as appropriate on host platforms with-

out KVM, or remove it to try under QEMU's TCG software emulation.

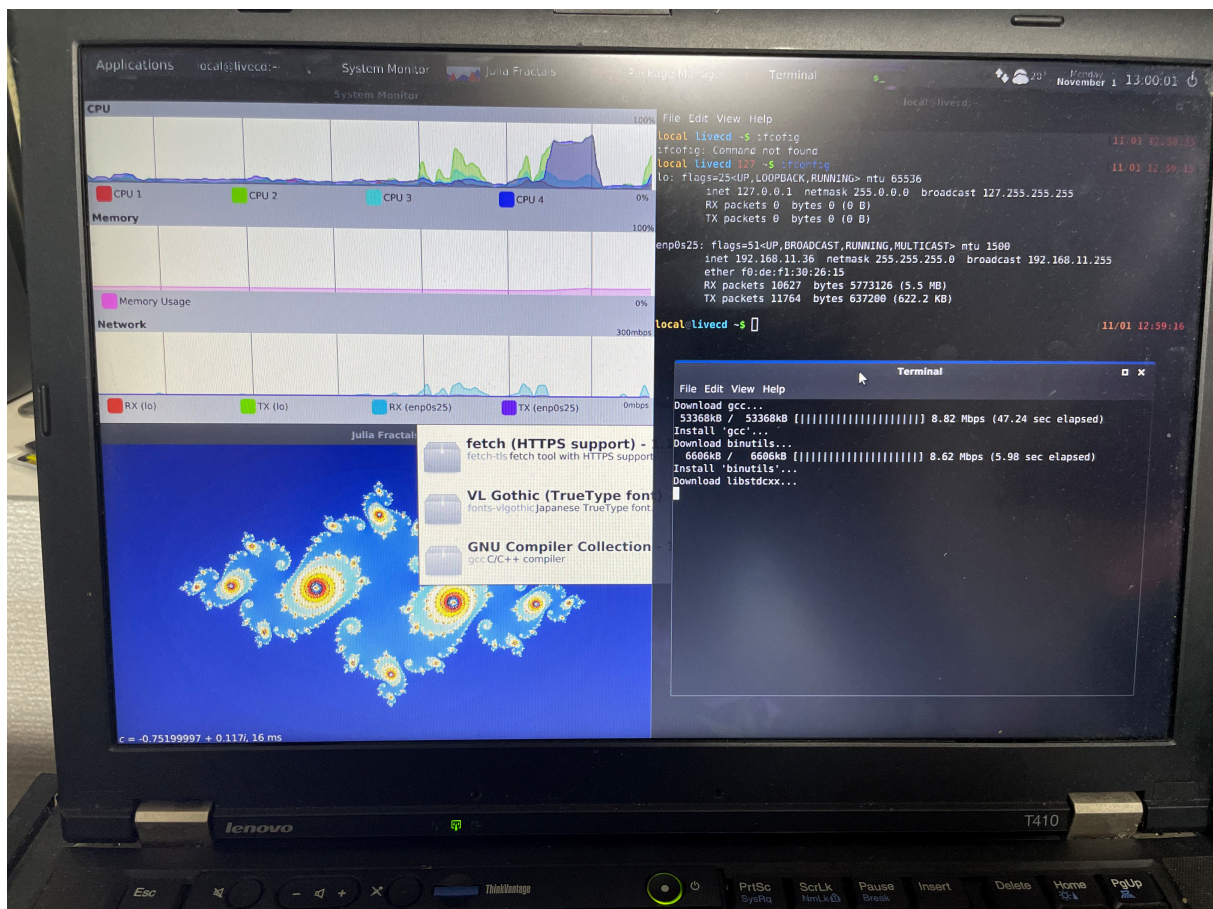
Note that QEMU command line options are not stable and these flags may produce warnings in newer versions.

The option `-M q35` will replace the PIIX chipset emulation with a newer one, which has the side effect of switching the IDE controller for a SATA one. This can result in faster boot times at the expense of ToaruOS not being able to read its own CD at runtime until I get around to finishing my AHCI driver.

Other

ToaruOS has been successfully tested on real hardware. If the native BIOS or EFI loaders fail to function, try booting with Grub. ToaruOS complies with the “Multiboot” and “Multiboot 2” specs so it may be loaded with either the `multiboot` or `multiboot2` commands as follows:

```
1 multiboot2 /path/to/misaka-kernel root=/dev/ram0 migrate vid=auto start
  =live-session
2 module2 /path/to/ramdisk.igz
3 set gfxpayload=keep
```



ToaruOS running natively from a USB stick on a ThinkPad T410.

License

All first-party parts of ToaruOS are made available under the terms of the University of Illinois / NCSA License, which is a BSD-style permissive license. Unless otherwise specified, this is the original and only license for all files in this repository - just because a file does not have a copyright header does not mean it isn't under this license. ToaruOS is intended as an educational reference, and I encourage the use of my code, but please be sure you follow the requirements of the license. You may redistribute code under the NCSA license, as well as make modifications to the code and sublicense it under other terms (such as the GPL, or a proprietary license), but you must always include the copyright notice specified in the license as well as make the full text of the license (it's only a couple paragraphs) available to end-users.

While most of ToaruOS is written entirely by myself, be sure to include other authors where relevant, such as with Mike's audio subsystem or Dale's string functions.

Some components of ToaruOS, such as Kuroko or bim have different but compatible terms.

Community

Mirrors

ToaruOS is regularly mirrored to multiple Git hosting sites.

- Gitlab: [toaruos/toaruos](#)
- GitHub: [klange/toaruos](#)
- Bitbucket: [klange/toaruos](#)
- ToaruOS.org: [klange/toaruos](#)

IRC

#[toaruos](#) on Libera ([irc.libera.chat](#))

FAQs

Is ToaruOS self-hosting?

Individual applications and libraries can be built by installing the [build-essential](#) metapackage from the repository, which will pull in [gcc](#) and [binutils](#). Sources are available in the [/src](#) direc-

tory on the live CD in a similar layout to this repository, and the `auto-dep.krk` utility script is also available.

For building ramdisks, finalized kernels, or CD images, some components are currently unavailable. In particular, the build script for ramdisks is still written in Python and depends on its `tarfile` module and `zlib` support. Previously, with a capable compiler toolchain, ToaruOS 1.x was able to build its own kernel, userspace, libraries, and bootloader, and turn these into a working ISO CD image through a Python script that performed a similar function to the Makefile.

ToaruOS is not currently capable of building most of its ports, due to a lack of a proper POSIX shell and Make implementation. These are eventual goals of the project.

Is ToaruOS a Linux distribution?

No, not at all. There is no code from Linux anywhere in ToaruOS, nor were Linux sources used as a reference material.

ToaruOS is a completely independent project, and all code in this repository - which is the entire code-base of the operating system, including its kernel, bootloaders, libraries, and applications - is original, written by myself and a handful of contributors over the course of ten years. The complete source history, going back to when ToaruOS was nothing more than a baremetal “hello world” can be tracked through this git repository.

When you say “complete”...

ToaruOS is complete in the sense that it covers the whole range of functionality for an OS: It is not “just a kernel” or “just a userspace”.

ToaruOS is *not* complete in the sense of being “done”.

Is ToaruOS POSIX-compliant?

While I aim to support POSIX interfaces well enough for software to be ported, strict implementation of the standard is not a major goal of the OS, and full compliance may even be undesirable.

Are contributions accepted?

ToaruOS is a personal project, not a community project. Contributions in the form of code should be discussed in advance. Ports and other work outside of the repo, however, are a great way to help out.

You can also help by contributing to Kuroko - which is part of why it's kept as a separate repository.