dtee

Simon Arlott

Apr 21, 2024

CONTENTS

CHAPTER ONE

DESCRIPTION

Run a program with standard output and standard error copied to files while maintaining the original standard output and standard error as normal.

CHAPTER TWO

PURPOSE

To run programs from cron, suppressing all output unless the process outputs an error message or has a non-zero exit status whereupon the original output will be written as normal and the exit code will be appended to standard error.

It will do this by default when invoked as `cronty`, providing an alternative to `cronic` but without splitting up the output.

(Also, to do `tee(1)` with standard output and standard error at the same time.)

CHAPTER

THREE

CONTENTS

3.1 Architecture

3.1.1 Background

Commands on Unix systems output messages to two logical streams: standard output (for normal messages) and standard error (for error messages). In the normal execution of the command there may be messages of both types being output and they are usually related so it's important that they appear in the correct order. This is normally achieved by having the file descriptors of both streams be the same underlying destination (a terminal, pipe to another process, a file). There is no distinction between streams by the operating system when the message is written.

3.1.2 Problem

Commands are not always run separately by a user on a terminal. They may be run from a script or unattended from `cron(8)`.

When this happens it can be useful to know if the process wrote any error messages, so typically the file descriptors for standard output and standard error would be different destinations (two separate log files). The script can then easily distinguish between normal messages and error messages.

This works well to determine the outcome of the command and examine any messages it outputs. A problem arises when it is necessary to provide the original output of the command to a user. It is not possible to guarantee the reading of messages from two file descriptors in the correct order and there is no assistance provided by the operating system for doing this.

The output from commands can be confusing if the messages are no longer in the original order. Splitting the output up into two blocks (normal and error) is not helpful.

Workarounds

One option has been to use `LD_PRELOAD` to modify the behaviour of the process and identify the destination stream as the messages are being written. This is error-prone because there are lots of different library functions for outputting to standard streams as well as functions within the C library that may bypass their external API and output messages directly.

Processes may also write directly to the file descriptors using system calls or their executables may be statically linked, preventing preloading from working. They may have multiple threads to complicate the manipulation of messages. For security reasons it's not possible to preload libraries into setuid executables using `LD_PRELOAD` so this option doesn't work for those commands.

3.1.3 Solution

Splitting of standard output and standard error while retaining the order of output can be performed using three `unix(7)` datagram sockets. A single input socket is used (so that messages can be read in order) and two output sockets are connected to the same input socket (so that they share the same reliable ordered buffer).

The source address of each message is provided by the operating system on every read so it is possible to identify which output stream was used by binding to different paths for each stream.

Alternatives

A new flag `O_SYNCPIPE` for `pipe2(2)` that creates a pair of pipes (4 file descriptors in total) where the read order of data is linked across the pair. This would have behaviour like `O_DIRECT` when transitioning from one stream to the other and only one of the pipes would be available for reading at any one time.

3.2 Dependencies

The following tools and libraries are required as part of the build process, to run dtee or to produce documentation.

3.2.1 Build

Compile

- Boost 1.56+ (1.82+ preferred)
- Clang 5+ or GNU GCC 5.3+
- GNU gettext
- GNU Make 3.80+ (optional)
- Meson 0.63+
  - Ninja
  - Python 3

Static Analysis

- Clang Static Analyzer
- Cppcheck

Test

- GNU Bash 4.1+
- GNU Core Utilities
- GNU Diff Utilities
- GNU Find Utilities
- GNU GCC gcov
- lcov

3.2.2 Runtime

- Boost
- GNU libintl
- UNIX domain sockets

3.2.3 Documentation

- Sphinx 4.1.0+

3.3 Limitations

Datagram sockets can only process writes as individual packets with a maximum packet size. Therefore, if the program being run attempts to `write(2)` more than this size in one call the write will fail and that part of the output will be lost.

This is not usually a problem because the default socket buffer size is usually much higher than the size programs typically write with. For safety, the socket buffer size will be increased to at least `PIPE_BUF` and `BUFSIZ` if the default is smaller than these values.

Writes to the socket (on Linux or GNU Hurd) will block until there is capacity available in the socket buffer. If the process uses `sendfile(2)` then (on Linux) the writes occur in `PIPE_BUF` sized chunks so it works as normal, but why are you using an interactive program that outputs such large quantities of data?

It is not possible to open `/dev/stdout` or `/dev/stderr` because they are sockets. No program would need to do this but like the use of `/dev/stdin` it may be desirable in scripts to work around limitations in file descriptor handling (however this is unlikely for an interactive program).

For more details read the *architecture* document.

3.3.1 BSD Operating Systems

Writes to the socket do not block when the receive buffer of the peer socket is full. The default socket receive buffer is quite small so it will be raised which avoids problems some of the time but messages are likely to be lost from programs that write large amounts of data very quickly or do so inefficiently (1 byte at a time).

On some versions of some operating systems the communication will be too unreliable for any kind of use and may or may not report errors.

3.3.2 GNU Hurd

Does not currently have support for returning addresses of Unix sockets, so none of the output works. It may be possible to implement custom pipe-like objects with three file descriptors in user space.

Writes larger than the page size (4KB) are truncated and there’s no way to increase the size of the socket buffer.

3.3.3 Cygwin

Performs as well as Linux but the maximum amount of data that can be streamed quickly is limited by the size of the socket buffer (which will will be raised to 2MB).

There are security issues because the underlying implementation of Unix sockets is a UDP socket on localhost. This presents an opportunity for another process to bind the same port after `dtee` or the command being run exits, which will allow it to:

- Write additional input for `dtee` after the child process has exited (until `waitpid(2)` is processed).
- Read output from child processes if the program being run forks into the background (causing `dtee` to exit).
- Read output from child processes if `dtee` is killed.

3.4 Build and install

The meson build system is used to build and install dtee:

```
meson build/release                # configure dtee build
ninja -C build/release             # compile
ninja -C build/release test        # run the tests
ninja -C build/release install     # install to default locations
```

A `Makefile` that calls `meson` and `ninja` is provided for convenience.

See *the list of dependencies* for more information on build, test and runtime requirements.

3.4.1 Cygwin

The build and test process makes use of symbolic links. If the repository is cloned outside of Cygwin these will not be present unless unprivileged symbolic links are enabled and all environments are configured to use native symbolic links.

Some of the tests cannot be run from an elevated process and will be skipped.

3.5 Manual page

3.5.1 Synopsis

dtee [*option*]… <*command*> [*arguments*]…

cronty [*option*]… <*command*> [*arguments*]…

3.5.2 Description

Run *command* with standard output and standard error copied to files while maintaining the original standard output and standard error as normal.

3.5.3 Options

Output files

Standard streams can be written to any number of specified files, in addition to normal output. Output is not line buffered.

-o <filename>, --out-append=<filename> Append standard output to *filename*, creating the file if it does not exist.

-O <filename>, --out-overwrite=<filename> Copy standard output to *filename*, truncating and overwriting existing content.

-e <filename>, --err-append=<filename> Append standard error to *filename*, creating the file if it does not exist.

-E <filename>, --err-overwrite=<filename> Copy standard error to *filename*, truncating and overwriting existing content.

-c <filename>, --combined-append=<filename> Append standard output and standard error to *filename*, creating the file if it does not exist.

-C <filename>, --combined-overwrite=<filename> Copy standard output and standard error to *filename*, truncating and overwriting existing content.

General options

-i, --ignore-interrupts Ignore keyboard interrupt signals (SIGINT). This does not prevent the command being executed (and other processes in the same progress group) from receiving the signal.

-q, --cron Operate in cron mode (this is implied when invoked as **cronty**). Suppresses all output unless the process outputs an error message or has a non-zero exit status whereupon the original output will be written as normal and the exit code will be appended to standard error.

Miscellaneous

-h, --help Display usage information and exit.

-V, --version Output version information and exit.

3.5.4 See also

Full documentation

CHAPTER

FOUR

RESOURCES

4.1 Change log

4.1.1 Unreleased

4.1.2 1.1.1 – 2024-04-20

Standard I/O checks, fixes for tests, and use of newer Boost.Asio (1.79 and 1.82) and C++17 features.

Changed

- Use the new Boost.Asio interface for setting the `SA_RESTART` flag on signal handlers when using Boost 1.82+.
- Disable "error location" in exception messages when building with Boost.Asio 1.79 because these exceptions are already caught and properly identified.
- Configure standard output and standard error to be blocking.
- Use `std::filesystem` from C++17 instead of `boost::filesystem`.
- Ensure that standard output and standard error exist.

Fixed

- Error message checks in tests when building with Boost 1.78.
- Locale handling in i18n tests when running on glibc 2.39.

4.1.3 1.1.0 – 2021-05-30

Bug fixes and support for internationalisation.

Added

- Internationalisation of output text messages.

Changed

- Normalise error messages.
- Make signals non-interrupting by adding the `SA_RESTART` flag after Boost.Asio sets the signal handler (chriskohlhoff/asio issue #646) instead of trying to wrap calls to `sigaction(2)`.
- Trivial performance improvements.
- Improved robustness when signals are received between forking a child process and executing the command.
- Errors writing to additional output files will always be reflected in the exit status even in cron mode.

Fixed

- Race condition between the command immediately exiting and being ready to handle `SIGCHLD`.
- Handle errors when the temporary directory name is too long or unusable.

4.1.4 1.0.1 – 2018-12-22

Update to avoid causing a trivial memory leak in Boost.

Fixed

- Memory leak in Boost program_options resulting from differing `boost::smart_ptr` implementations (boost-org/program_options issue #70).

4.1.5 1.0.0 – 2018-12-09

First stable release.

Added

- Best effort support on Darwin (macOS).
- Best effort support on Cygwin.

Fixed

- Invalid usage messages now use standard error instead of standard output.
- Check build version matches the release version.

4.1.6 0.0.1 – 2018-11-11

Update to allow improvements in packaging.

Fixed

- Infinite loop in the test scripts if check variables are undefined (this is unlikely).
- Support for unity builds when `-Wshadow` is used.

4.1.7 0.0.0 – 2018-11-09

Initial development release for packaging.

Added

- Full support on Linux.
- Best effort support on FreeBSD and OpenBSD.
- Basic support on NetBSD and DragonFlyBSD.
- Compiles on GNU Hurd (but doesn't work).
- Comprehensive automated tests of all functionality.

4.2 Packages

Source packages for Linux distributions are kept in the dtee-package repository.

4.2.1 Debian binary packages

Packages are available for:

Debian

Follow the instructions for your release. If you are using a newer release than the ones listed then use the builds for the most recent prior version.

Debian 12 (bookworm)

Run the following commands to install the `repository public key`, APT data source `dtee-debian-bookworm.list` and then `dtee`:

```
wget https://dtee.bin.uuid.uk/debian/repo-key.gpg \
    -O /etc/apt/keyrings/dtee.gpg

echo "deb [signed-by=/etc/apt/keyrings/dtee.gpg]" \
    "https://dtee.bin.uuid.uk/debian/ bookworm main" \
```

(continues on next page)

(continued from previous page)

```
    >/etc/apt/sources.list.d/dtee.list

apt update
apt install dtee
```

Debian 11 (bullseye)

Run the following commands to install the `repository public key`, APT data source `dtee-debian-bullseye.list` and then `dtee`:

```
mkdir -m 0755 -p /etc/apt/keyrings

wget https://dtee.bin.uuid.uk/debian/repo-key.gpg \
    -O /etc/apt/keyrings/dtee.gpg

echo "deb [signed-by=/etc/apt/keyrings/dtee.gpg]" \
    "https://dtee.bin.uuid.uk/debian/ bullseye main" \
    >/etc/apt/sources.list.d/dtee.list

apt update
apt install dtee
```

Debian 10 (buster)

Run the following commands to install the `repository public key`, APT data source `dtee-debian-buster.list` and then `dtee`:

```
mkdir -m 0755 -p /etc/apt/keyrings

wget https://dtee.bin.uuid.uk/debian/repo-key.gpg \
    -O /etc/apt/keyrings/dtee.gpg

echo "deb [signed-by=/etc/apt/keyrings/dtee.gpg]" \
    "https://dtee.bin.uuid.uk/debian/ buster main" \
    >/etc/apt/sources.list.d/dtee.list

apt update
apt install dtee
```

Ubuntu

Follow the instructions for your release. If you are using a newer release than the ones listed then use the builds for the most recent prior version.

Ubuntu 22.04 LTS (Jammy Jellyfish)

Run the following commands to install the `repository public key`, APT data source `dtee-ubuntu-jammy.list` and then `dtee`:

```
wget https://dtee.bin.uuid.uk/ubuntu/repo-key.gpg \
    -O /etc/apt/keyrings/dtee.gpg

echo "deb [signed-by=/etc/apt/keyrings/dtee.gpg]" \
    "https://dtee.bin.uuid.uk/ubuntu/ jammy main" \
    >/etc/apt/sources.list.d/dtee.list

apt update
apt install dtee
```

Ubuntu 20.04 LTS (Focal Fossa)

Run the following commands to install the `repository public key`, APT data source `dtee-ubuntu-focal.list` and then `dtee`:

```
mkdir -m 0755 -p /etc/apt/keyrings

wget https://dtee.bin.uuid.uk/ubuntu/repo-key.gpg \
    -O /etc/apt/keyrings/dtee.gpg

echo "deb [signed-by=/etc/apt/keyrings/dtee.gpg]" \
    "https://dtee.bin.uuid.uk/ubuntu/ focal main" \
    >/etc/apt/sources.list.d/dtee.list

apt update
apt install dtee
```

4.2.2 RPM packages

Packages are available for:

Fedora

Supported versions: 38 and 39.

Save the repository configuration file `dtee-fedora.repo` to `/etc/yum.repos.d/dtee-fedora.repo`.

Run the following command:

```
yum install dtee
```

Yum will prompt to confirm the repository `public key`:

```
4784 9A12 DAF9 BD2A F550 5FBB 4FF8 86F3 1820 6BD9
```

Red Hat Enterprise Linux

Yum will prompt to confirm the repository `public key`:

```
4784 9A12 DAF9 BD2A F550 5FBB 4FF8 86F3 1820 6BD9
```

Red Hat Enterprise Linux 9

Save the repository configuration file `dtee-rhel9.repo` to `/etc/yum.repos.d/dtee-rhel9.repo`.

Run the following command:

```
yum install dtee
```

Red Hat Enterprise Linux 8

Save the repository configuration file `dtee-rhel8.repo` to `/etc/yum.repos.d/dtee-rhel8.repo`.

Run the following command:

```
yum install dtee
```

Red Hat Enterprise Linux 7

Save the repository configuration file `dtee-rhel7.repo` to `/etc/yum.repos.d/dtee-rhel7.repo`.

Run the following command:

```
yum install dtee
```

Red Hat Enterprise Linux 6

Boost 1.60 from the Red Hat Software Collections is required.

Save the repository configuration file `dtee-rhel6.repo` to `/etc/yum.repos.d/dtee-rhel6.repo`.

Run the following commands:

```
subscription-manager repos --enable "rhel-*-rhscl-6-rpms"
yum install dtee
```

4.2.3 Arch Linux package

A package for dtee is available to build from the Arch User Repository.

CHAPTER

FIVE

PRONUNCIATION

dtee	/di.ti/
cronty	/krn.ti/