GSoC 2018 Project Proposal

Name:	Sukrit Bhatnagar
Location:	Mumbai, India (UTC +5:30)
Email ID:	skrtbhtngr@gmail.com, sukrit@cse.iitb.ac.in
IRC Nickname:	skrtbhtngr
Phone:	+91 766 520 6152
Academics:	M.Tech. (RA) in Computer Science and Engineering at Indian Institute of Technology Bombay

Project: Automatic freeing of resources

Summary: Implement `__attribute__((cleanup))` for libvirt

Mentor: Pavel Hrdina

Suggested By: Daniel Berrangé

Abstract

Writing cleanup code for every variable in large-scale open-source projects makes the code difficult to comprehend. The cleanup tasks like releasing resources are necessary and without them, there may be leaks. GNU C provides a variable attribute `cleanup` that runs a specified function automatically when the associated variable goes out of scope. The aim of this project is to incorporate this cleanup functionality into the libvirt code base. Another aspect of this project will be to ensure that adding this functionality does not change the intended behaviour of the code.

The deliverable will be an integration of cleanup code into the libvirt code base which does not interfere with the main logic, making the code more concise and comprehensible while maintaining intended functionality.

About this attribute

`cleanup` is one of many variable attributes provided by GNU C and is supported since GCC version 3.3.1 (released in Aug 2003) [2]. This attribute is already being used by many open source projects (such as libguestfs) and many are planning to use it in the near future. It performs the task of automatically running a specified function when the associated local variable goes out of scope. The functions which are supported have some characteristics:

- They cannot return any value.
- They can take only one parameter which must be a pointer to the type of associated variable (or can be a generic void pointer!).

Some caveats of using this attribute:

- It can be applied only to the local (automatic) variables, not to global or static variables or even function parameters.
- If a variable is uninitialized and the function returns before its initialization, the cleanup function might behave unexpectedly or give errors.
- The corresponding cleanup function for a local variable does not run if the enclosing function exits abnormally, such as on exit() or non-local jumps.
- If a pointer variable is to be returned from any function, it cannot be declared with this attribute, as its value might be expected by the caller.
- Since the cleanup function cannot return any value, it cannot be used to report any error that might occur during cleanup.
- Many variables have to be annotated with this attribute.
  `#define autoclean __attribute__((__cleanup__(anyfunc)))`
  Even when we `#define` an identifier to be used in place of it (like above), it might not be obvious to the reader of the code, and may also lead to a more complex declaration part of a function.

Motivation

Currently, the `virCommandFree` function does the job of releasing the resources of a `virCommandPtr` by calling the `VIR_FREE` macro which eventually calls the `virFree` function. In functions such as `lxcContainerChild` in `src/lxc/lxc_container.c`, a jump to `cleanup` label invokes these functions. Goto jumps like these are discouraged and can be avoided by using this attribute. This is one of many examples where the need for this attribute is reflected. In addition to eliminating the need for goto jumps, this can help in separating cleanup logic from the main logic and thus improving code quality.

Overview

Functions:

There are broadly two categories of *existing* functions for cleanup in the code:

- Functions which take a pointer to the type to free and do not return any value; they are valid cleanup functions for the attribute. E.g. `virCommandFree`, `virConfFreeList` and `virCgroupFree`. They can be reused without any modifications.
- Functions whose return type and/or parameter list differ from that of a valid cleanup function. E.g. `virFileClose` and `virFileFclose`. These functions require some additional mechanism to pass extra values and/or return a value back. One way to go is to set some global variable(s) in a separate source file (like errno variable from errno.h) and then access that global variable in caller/callee function.

The proposed cleanup functions can perform tasks such as automatically freeing allocated memory, clearing the memory, closing file descriptors and unlocking mutexes.

Some examples of existing functions whose role can be played by cleanup functions:

“free” functions:

- `virConfFreeList` and `virConfFreeValue` functions defined in `src/util/virconf.c`
- `virCommandFree` function defined in `src/util/vircommand.c`
- `virFreeError` function defined in `src/util/virerror.c`

“dispose” functions:

- `virConnectDispose`, `virDomainDispose` etc. defined in `src/datatypes.c`

“close” functions:

- `virFileClose`, `virFileFclose` defined in `src/util/virfile.c`

Also, for the case where a function exits unusually, this attribute will not work. Either some function must be called to explicitly free resources or necessary logic be embedded sooner in the code.

Variables:

There are many occurrences of local variables, where `VIR_FREE` / `VIR_DISPOSE*` / `VIR_*_CLOSE` / `VIR_*_FCLOSE` is explicitly invoked to free allocated resources. These variables are the prime candidates for this attribute.
When a variable is considered for this attribute, we must ensure that is not returned by the function. Also, we have to analyze if an uninitialized variable should be initialized at declaration just to be used with this attribute, so that it is cleaned up even if the function returns prematurely before its actual initialization.

There are many ways in which a variable can be made to use this attribute:

- Annotate the declaration of a variable (directly) in the function code with this attribute. Basically, a macro like the following can be defined.
  `#define autoclean __attribute__((__cleanup__(anyfunc)))`
  Then in the declaration of a variable, `autoclean` can be added.
  This has the drawback of creating clutter in the code, something which we are trying to avoid in the first place.

- Create a new type out of an existing type which will include this attribute. If a data type is created using typedef, we can create a macro for that type which includes the cleanup attribute and use this macro for type declarations, so we do not need to annotate each declaration of a variable of that type. For example:

  ```
  typedef virCommand *virCommandPtrType;
  #define virCommandPtr autoclean virCommandAutoPtr
  ```

  Another way to explore is using macros themselves to declare any variables that need automatic cleaning. For example, instead of:

  ```
  virCommandPtr cmd = virCommandNewArgs(args);
  ```

  we can have:

  ```
  #define CREATE_CMD_PTR(var, val) \
       autoclean virCommandPtr var = (val);
  ...
  CREATE_CMD_PTR(cmd, virCommandNewArgs(args))
  ```

  The latter will ensure that we do not forget to initialize variables, and we will get no problems in cleanup later.

A cleanup function for each data type/structure to be cleaned up automatically is needed, if not already available. For native data types, `virFree` and `virDispose` are already available. But each complex data type (struct variable) will have its own special cleanup code.
Some examples of the important structures which need a special cleanup function are `virConfPtr`, `virConfValuePtr`, `virErrorPtr` and `virCommand`.

In addition to the changes in the code, we need some syntax check rules to ensure that the semantics are maintained and to prevent any warning messages that compiler may generate on misuse of this attribute. Some of the new syntax-check rules might enforce these rules:

- No static or global variables must use this attribute.
- All variables declared with this attribute must be initialized (before any attempt to leave the function can be made).

Timeline

Apr 23 - May 14 (Community Bonding Period)

- Read the relevant parts of libvirt API documentation. Get well-acquainted with the code structure and development tools used.
- Ask the mentor(s) for suggestions and guidance. Discuss the general design for implementing the attribute with the libvirt community.

May 14 - May 28 [2 weeks]

- Go through the code to find functions where this attribute can be used, and will benefit from it. Especially, all goto jumps to `error`, `cleanup` and other similar labels must be considered.
- Check to see if any additional functionality is required in the function's context in addition to cleanup.

May 28 - Jun 18 [3 weeks] : Jun 11 - Jun 15 (First Evaluation)

- Create a suitable design for adding the functionality in the code. The design will specify which existing files to modify, what new files to create and how existing code interacts with the new code that will be added.
- Have discussions with the mentor and the community about the design as this will involve large amounts of changes in the code.
- Implement the necessary cleanup functions and modify the existing functions if needed. Add suitable comments.
- Make checks regularly to see if anything breaks and debug the code for any errors reported.
- Send patches continuously to the mailing list for feedback and suggestions and make necessary changes according to the feedback received.
- **Deliverable for the evaluation:** code design and implementation of cleanup functions ready to be used with the attribute.

Jun 18 - Jul 9 [3 weeks]

- Annotate the declarations of variables (one type at a time) over parts of the code with the attribute. This can be done by making changes in one module/hypervisor/driver implementation at a time.
- Make checks regularly to see if anything breaks and debug the code for any errors reported. The checks can be made when a complete source file is modified, or even when a subset of functions are. Since libvirt has active contributors, it is better to make check after rebasing when any other developer has made a commit.
- Send patches continuously to the mailing list for feedback and suggestions and make necessary changes according to the feedback received.

Jul 9 - Jul 16 [1 week] : Jul 9 - Jul 13 (Second Evaluation)

- **Deliverable for the evaluation:** successful integration of this attribute into the code which passes make checks.
- Write new tests, if necessary, to ensure that the cleanup logic is implemented correctly and is working as expected.
- Add syntax-check rules to check if the attribute is used correctly. This will ensure that any future additions or modifications to the code base adhere to the attribute's semantics.

Jul 9 - Jul 23 [1 week]

- After all changes are made, scan the code for potential improvements on code quality, readability and length.
- Add helpful comments wherever necessary.

Jul 23 - Jul 30 [1 week]

- Write blog posts about any significant changes made which are accepted.
- Document all the changes made.

Jul 30 - Aug 6 [1 week]

- This week is kept as buffer to cope up with any unexpected event.

Final Deliverable

- Integration of cleanup attribute into libvirt code which results in enhanced code quality and maintains the intended functionality.
- New syntax-check rules and documentation which will help other developers contributing to the code in maintaining the semantics of the newly implemented attribute.

Note: The proposed timeline is open to any changes suggested by the mentor(s).

About Me

I am Sukrit Bhatnagar, a postgraduate student at IIT Bombay. My areas of interest are System Virtualization, Linux Kernel programming, NLP, and Deep Learning. This is my GitHub profile.

I also work as a System Administrator for my department (Research Assistantship). I have experience in administering Linux-based systems and servers.

I had attained the title of RHCA (Red Hat Certified Architect: 130-114-002) in May 2015 by completing several exams related to administering server related to Red Hat Enterprise Linux.

To dive into the libvirt code base and have a small upstream contribution, I have sent a patch to the libvirt mailing list to add comma escaping using `virQEMUBuildBufferEscapeComma`.

Other commitments

I have a commitment of 2-3 hours per day (Mon-Fri) in the summers during Coding Period for system administration work. Other than that, I am completely available for GSoC and can dedicate 40 hours a week to it.

Why choose me?

- I am interested in contributing to libvirt
- I am proficient in C and it is my primary language of choice
- I am a quick learner, especially for large projects in C
- I am inclined towards system/OS - level virtualization programming

References

1. https://wiki.libvirt.org/page/Google_Summer_of_Code_Ideas#Automatic_freeing_of_memory
2. https://gcc.gnu.org/onlinedocs/gcc-3.3.1/gcc/Variable-Attributes.html#Variable-Attributes
3. Libvirt mailing list discussions: https://www.redhat.com/archives/libvir-list/