Kernel Dev Intro and Better Code

Personal Experience Sharing by Yixin Shen

Disclaimer

- The views expressed here are purely based on my personal experiences and thoughts. Please regard them as for reference only.
- There could be inaccuracies or omissions in my content, so constructive feedback and corrections are welcome.
- Kernel Part Post: https://bobankh.com/posts/contribute-kernel/
- Coding Part Post: https://bobankh.com/posts/better-code/

Table of **contents**

01

Dev Env

Setup the kernel

Virtual Machine

- Avoid mess up
- Choice: Vmware, VirtualBox, qemu, Firecracker, etc.

- Recommendation:
  - Easy-to-Use
  - Flexible

Installation

- Very Convenient
- Script: vagrant-libvirt/vagrant-libvirt-qa/scripts/install.bash
- Grant privileges:

```
sudo usermod -aG kvm "$USER"
sudo usermod -aG libvirt "$USER"
```

- Vagrantfile
  - Box: os-image
  - Plugin
  - Provision

Usage

```
    destroy         stops and deletes all traces of the vagrant machine
    global-status   outputs status Vagrant environments for this user
    halt            stops the vagrant machine
    help            shows the help for a subcommand
    init            initializes a new Vagrant environment by creating a Vagrantfile
    login
    package         packages a running vagrant environment into a box
    plugin          manages plugins: install, uninstall, update, etc.
    port            displays information about guest port mappings
    powershell      connects to machine via powershell remoting
    provision       provisions the vagrant machine
    push            deploys code in this environment to a configured destination
    rdp             connects to machine via RDP
    reload          restarts vagrant machine, loads new Vagrantfile configuration
    resume          resume a suspended vagrant machine
    serve           start Vagrant server
    snapshot        manages snapshots: saving, restoring, etc.
    ssh             connects to machine via SSH
    ssh-config      outputs OpenSSH valid configuration to connect to the machine
    status          outputs status of the vagrant machine
    suspend         suspends the machine
    up              starts and provisions the vagrant environment
    upload          upload to machine via communicator
```

02

Make Patch

How to contribute to upstream

Where to Begin

- **/MAINTAINERS**
  - M: Maintainers
  - R: Reviewers
  - T: Source Tree
  - L: Mailing List
  - W: Web Page
  - S: Status
  - F: Files
  - …

```
BPF [GENERAL] (Safe Dynamic Programs and Tools)
M:      Alexei Starovoitov <ast@kernel.org>
M:      Daniel Borkmann <daniel@iogearbox.net>
M:      Andrii Nakryiko <andrii@kernel.org>
R:      Martin KaFai Lau <martin.lau@linux.dev>
R:      Song Liu <song@kernel.org>
R:      Yonghong Song <yhs@fb.com>
R:      John Fastabend <john.fastabend@gmail.com>
R:      KP Singh <kpsingh@kernel.org>
R:      Stanislav Fomichev <sdf@google.com>
R:      Hao Luo <haoluo@google.com>
R:      Jiri Olsa <jolsa@kernel.org>
L:      bpf@vger.kernel.org
S:      Supported
W:      https://bpf.io/
Q:      https://patchwork.kernel.org/project/netdevbpf
T:      git git://git.kernel.org/pub/scm/linux/kernel/
T:      git git://git.kernel.org/pub/scm/linux/kernel/
F:      Documentation/bpf/
F:      Documentation/networking/filter.rst
```

How to Create a Patch

- Single Patch

```
git format-patch --subject-prefix='PATCH' -i HEAD~
```

- Patch Series

```
git format-patch --cover-letter --subject-prefix='PATCH' -N
```

  - N+1 patches including cover letter

```
[...]
Subject: [PATCH 0/2] *** SUBJECT HERE ***

*** BLURB HERE ***

[...]
```

How to Create a Patch

- Modification to re-submit

```
[...]
Subject: [PATCH v3 bpf-next 0/2] *** SUBJECT HERE ***

*** BLURB HERE ***

v3:
 - xxxx
 - yyyy
v2:
 - zzzz
[...]
```

Before Submit

- Functionality Check
  - **Must** be compiled
  - **Must** pass all tests
- Style Check
  - scripts/checkpatch.pl
  - ERROR, WARNING, CHECK

How to Send a Patch

- Configure Email Client
  - Plain-Text
  - **git-email**

```
[sendemail]
        smtpencryption=tls
        smtpserver=smtp.gmail.com
        smtpuser=<youraccount>@gmail.com
        smtpserverport=587
[credential]
        helper = store
```

Where to Send a Patch

- scripts/get_maintainer.pl

```
Alexei Starovoitov <ast@kernel.org> (supporter:BPF [GENERAL]
(Safe Dynamic Programs and Tools),commit_signer:1/2=50%)
```

- Send **TO** the Mailing List of subsystem
- **CC** to some of maintainers and reviewers

```
git send-email --to <a@m.com> --cc <b@m.com> 00*.patch
```

- A little demo patch
- An awesome patch template:
  - “Just give me 1 month to write a thesis for this patch…”

03

Better Code

Principles, Patterns and Practices

Code X

- Principles:
  - high-level abstract ideas and philosophies
- Patterns:
  - mid-level reusable solutions to commonly occurring software design problems
- Practices:
  - specific, granular techniques and examples for coding

Principles

- Provide rules to eliminate bad solutions and pick a good one
- A set of high-level guidelines that help ensure clean, simple and readable code

CRISP

- Correct
- Readable
- Idiomatic: conventional
- Simple: directness, frugality
- Performant

High cohesion, Loose coupling

- Provide rules to eliminate bad solutions and pick a good one
- A set of high-level guidelines that help ensure clean, simple and readable code
- High cohesion: strongly related and focused
- Loose coupling: self-contained, not have strong dependence on other modules or components

High cohesion, Loose coupling

- Easier to understand
- More flexible
- More resilient
- Reusable
- Independent development
- Fault isolation

SOLID

- Single Responsibility Principle(SRP)
- Open-Closed Principle(OCP)
- Liskov Substitution Principle(LSP)
- Interface Segregation Principle(ISP)
- Dependency Inversion Principle(DIP)

KISS

- Keep It Simple, Stupid
- Simple designs are more readable, testable, and maintainable. They have fewer bugs and edge cases.

YAGNI

- You Aren't Gonna Need It
- Not adding extra features or optimizations that aren't essential based on current requirements
- First get something working, then make it optimal

DRY

- Don’t Repeat Yourself
- Aims to simplify your code and improve decoupling and reusability
- But if you find it more straightforward to understand by copying and pasting, then do so

SoC

- Separation of Concerns
- Detangling code so it can work in isolation
- DRY vs SoC: concepts
- Despite superficial similarities of code, we should focus on the inner concepts

Observability

- Logging: historical record of discrete events
- Tracing: the flow of a request through the system end-to-end
- Utilizes different log levels to gain optimal observability

Patterns

- Represent typical solutions to recurring challenges or template solutions to well-known problems

Inheritance or Composition

- Inheritance is about 'A-is-B'
- Composition is about 'A-has-B'
- Benefits of composition:
  - Loose coupling
  - Flexibility
  - Testability
  - Single Responsibility Principle
  - Dependency Inversion Principle
  - Readability
  - Cache-Friendliness(Data-Oriented-Design)

Test-Driven Development

- First implement a set of tests for every new feature being added to the software.
- These tests are expected to fail as the new feature has not been implemented yet.
- Once it is implemented, all previous tests and the new ones should pass.
- **Describe what you would like the software to be before developing new features.**
- **Tests are the best in-code documentation**

Test-Driven Development

- **But always remember:**
  - Tests can only prove that the code does what the test writer thought it should, and most of the time they don't even prove that.
  - A faulty or insufficient test is much worse than no test at all, because it _looks_ like you have tests.

Clean Boundaries: Learning Tests

- While we are not responsible for directly testing third-party libraries and frameworks, we should write tests that exercise our usage of them.
- To validate that our understanding and integration of a third-party API is correct
- Serve as monitors that can detect if and when a third-party update introduces unintended behavior changes

Clean Boundaries

- When you have a bunch of third-party APIs to use together, it is recommended to bundle them together and wrap them in a modular.
- Create a shim-layer to isolate between your code and third-party code.
- To achieve flexible adjustments and changes and encapsulating details

Adaptor Pattern

- Allows the interface of an existing class to be used as another interface.
- It is often used when the interface of an existing class is not compatible with what the client code requires.
- Client
- Target
- Adaptor

Factory Pattern

- Use factory instead of constructor
- Employs a separate method to handle object creation
- More flexible

Strategy Pattern

- Defines an interface common to all supported algorithms, making them interchangeable.
- Allows you to switch algorithms dynamically without needing to modify the *Context*.

Dependency Injection

- A class receives its dependencies from external sources rather than creating them itself.
- Increased flexibility and modularity, testability, reduced coupling between components.
- By decoupling objects from their dependencies, changes can be made more easily without affecting other parts of the system.

Practices

RAII

- Resource Acquisition Is Initialization
- Ties the lifetime of resources to the lifetime of the objects that own them.
- Handles resource cleanup in a scope-based, systematic manner
- When an object goes out of scope, its destructor is called, which frees any owned resources.

Refactor

- Extract Method/Extract Class
- Improve Code Reusability
- Reduce Hard-Code for Maintenance
- Enhance Extensibility
- Decoupling
- Layering
- DDD(Domain Driven Design)

Refactor

- Extract Method/Extract Class
  - Group related functionality together
  - Each method should do one clear thing
  - Extract classes to each be responsible for only one concept (Single Responsibility Principle)

Refactor

- Extract Method/Extract Class
- Improve Code Reusability
  - DRY
  - Eliminate duplication
  - Extract method/class
  - Encapsulate into entity classes
  - Replace inheritance with combination
  - Combine inheritance with template method

Refactor

- Extract Method/Extract Class
- Improve Code Reusability
- Reduce Hard-Code for Maintenance
  - Replace magic number with a symbolic constant
  - Reduce the risk of making mistakes when changing the value

Refactor

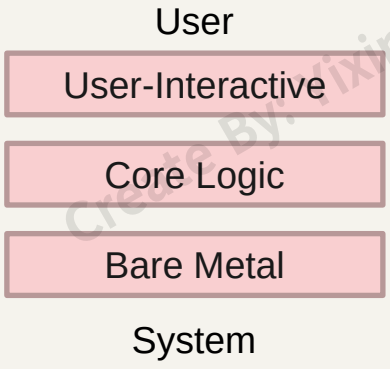
- Extract Method/Extract Class
- Improve Code Reusability
- Reduce Hard-Code for Maintenance
- Enhance Extensibility
  - Open-Closed Principle: We *Open for extension* but *Close for modification*.
  - Template method
  - AOP(Aspect Oriented Programming)

Refactor

- Extract Method/Extract Class
- Improve Code Reusability
- Reduce Hard-Code for Maintenance
- Enhance Extensibility
- Decoupling
  - More 'pluggable'
  - Leverage more design patterns to decouple our code

Refactor

- Extract Method/Extract Class
- Improve Code Reusability
- Reduce Hard-Code for Maintenance
- Enhance Extensibility
- Decoupling
- Layering
  - User-Interactive layer
  - Core Logic layer
  - Bare Metal layer



Refactor

- Extract Method/Extract Class
- Improve Code Reusability
- Reduce Hard-Code for Maintenance
- Enhance Extensibility
- Decoupling
- Layering
- DDD(Domain Driven Design)

Refactor

- Extract Method/Extract Class
- Improve Code Reusability
- Reduce Hard-Code for Maintenance
- Enhance Extensibility
- Decoupling
- Layering
- DDD(Domain Driven Design)

Documentation

- Explains how code works and why it was built that way
- Be Clear and Concise
- Examples and Snippets
- Provide Context
- Use Consistent Formatting and Style
- Document Edge Cases and Error Handling

04

Collaboration

Pull Request <= **PR** => Peer Review

Conventional Commit

<type>[optional scope]: <description>

[optional body]

[optional footer(s)]

Conventional Commit

- feat
- fix
- refactor
- docs
- test
- chore
- ci
- perf
- revert
- style

```
fix(api): prevent racing of requests

Introduce a request id and a reference
to latest request. Dismiss incoming
responses other than from latest
request.

Remove timeouts which were used to
mitigate the racing issue but are
obsolete now.

Reviewed-by: Z
Refs: #123
```

PR

Pull Request <= PR => Peer Review

Pull Request

- Inform other developers that you created a new branch, corresponding to a new version of your source code.
- Others can then see what are the differences and comment on them, eventually approving or declining the merge of your changes into the mainline.
- **Two heads are better than one.**
- The theory behind this is that by giving chance to others to look at your proposed changes, they can spot errors you missed before they go into production.

Peer Review

- A knowledge-sharing mechanism for other team members
- Avoid only single person understand some code pieces
- Reduce the number of bugs introduced to the mainline
- Reduce the accidental technical debt accrued

Peer Review

- A common scenario of PR

John is a developer and he worked for a full week adding a new feature to our product. He creates the PR request with 55 files and no one reviews it for a couple of days.

After the third time asking for help, he either receives one of the two types of feedback:

- 3 comments on the choice of variable names, a suggestion to use `forEach` instead of for loop and an LGTM at the end.
- A single comment describing how the feature/integration/abstraction is not correct and requires a major rewrite.

Peer Review

- A common scenario of PR

I Am Devloper
@iamdevloper

10 lines of code = 10 issues.

500 lines of code = "looks fine."

Code reviews.

5:58 PM · Nov 5, 2013

Peer Review

- A common scenario of PR
- The potential issues and their consequences:
  - Worked for a full week

Peer Review

- A common scenario of PR
- The potential issues and their consequences:
  - Worked for a full week
  - Creates the PR request with 55 files
    - PR should be concentrated
    - a review of 200–400 LOC over 60 to 90 minutes should yield 70–90% defect discovery
    - should not have more than 250 LOC to review

Peer Review

- A common scenario of PR
- The potential issues and their consequences:
  - Worked for a full week
  - Creates the PR request with 55 files
  - No one reviews it for a couple of days

Review Process

Modern Code Review: A Case Study at Google[ICSE-SEIP '18]:

- Creating
- Previewing
- Commenting
- Addressing Feedback
- Approving

PR: Reviewer

- Focus review on clarity and correctness first, then standards
- Fundament your feedback

Bad Review: "This code is bad. Why are you doing a linear search?"

Good Review: "This code block could be optimized by using a binary search instead of a linear one. Applied this would improve performance when searching large data sets."

- Don’t be afraid to ask questions

PR: Author

- Keep your changes concentrated and small
- Review your code before submitting
- A well-organized PR description
  - A Clear Title
  - A Detailed Description
  - Always review your changes first
  - Provide references

Thanks

https://bobankh.com/slides/kdev-coding.pdf