Poll Me

Het Thakkar, Neelkanth Tripathi, Divyank Gupta, Asha Khatri, and Lorenz

North Carolina State University

September 30, 2021

Abstract

This report explains how we as a group have implemented the Linux Kernel Best Practices in our project name Poll Me , We will be explaining how various attribute of the rubric match to the Linux Kernel Best practices.

1 Distributed Development Model

A distributed model is the best way of taking the development ahead, assigning different portions of the kernel (such as networking, wireless, device drivers, etc.) to different individuals, based on their familiarity with the area. This enabled seamless code review and integration across the thousands of areas in the kernel without any compromise in kernel stability.[1]

We demonstrated Distributed Development Model principle in Poll Me project by following these practices: The group members attended tutorial sessions, all the important decision were made through an unanimous vote and such meetings had a round robin order in which members would speak. Each meeting had a leader who would conduct the session and allow each member to discuss their idea, in addition to this the leader who would conduct such meetings was changed every meeting. Workload was spread among all team members and each member contributed. There are commits by each member of the team. There are docs for backend, project architecture which explains the working of various components of the app. Also a video is attached to the github repo to show how our app works.We have used the MIT license and have a number of different badges like DOI badge, build passing badge.

2 Consensus Oriented Model

The Linux kernel community strictly adheres to the consensus-oriented model, which states that a proposed change cannot be integrated into the code base as long as a respected developer is opposed to it. Although this might frustrate individual developers, such a practice ensures that the integrity of the kernel is not tampered with; no single group can make changes to the code base at the expense of the other groups. Consequently, the kernel's code base remains as flexible and scalable as always.[1]

We incorporated Consensus Oriented Model in Poll Me project by following these practise: All discussion's were held by the consensus of all team members and the issues were discussed before they were terminated , There exists a Chat channel which is active and was used heavily to discuss various essential things about the project. There exists files like CONTRIBUTING.md and CODEOFCONDUCT.md which discusses the ideology behind how our code was written and how it can be extended and what future work can be done in order to extend it.

3 Zero Internal Boundaries

Developers generally work on specific parts of the kernel; however, this does not prevent them from making changes to any other part as long as the changes are justifiable. This practice ensures that problems are fixed where they originate rather than making way for multiple workarounds, which are always a bad news for kernel stability. Moreover, it also gives the developers a wider view of the kernel as a whole.[1]

The whole team used the same tools like Visual Studio Code, AWS, Prisma, Netlify and all the code was being pushed to GitHub from the starting so that each one can pull it and work accordingly.Config and ENV files were shared with each member so that they can run the code properly on their local machine.

4 No Regression Rules

The kernel developer community continually strives to upgrade the kernel code base, but not at the cost of quality. This is why they follow the no-regressions rule, which states that if a given kernel works in a specific setting, all the subsequent kernels must work there too. However, if

the system does end up affected by regression, kernel developers waste no time in addressing the issue and getting the system back to its original state.[1]

Thus, we made sure that each member of the group had utmost priority to make sure that the code worked as planned on system's of each member of the group.

5 Short Release Cycles

In the early days of kernel development, major releases would come once every few years. This, however, led to several glitches and problems in the development cycle. For instance, long release cycles meant that vast chunks of code had to be integrated at once, which proved to be rather inefficient. It also translated into a lot of pressure for the developers to integrate features in the upcoming release even if they were not completely stable or ready. Moreover, delay in releasing new features was frustrating for users and distributors alike.[1]

That's why, the Linux kernel development community decided to switch to short release cycles, which addressed all the issues with long release cycles. New code is immediately integrated into a stable release. Plus, continually integrating new code allows for introducing fundamental changes into the code base without causing major disruptions. Besides, there is no pressure on the developers, given the short span between two release cycles.

In this project we made sure that all members commit and push their code in small steps after deciding the objective for each piece of code uploaded and this is how we maintained consistency and made sure everyone understood the code and what the team wanted to achieve at every stage.

6 References

[1] https://medium.com/@Packt$_{P}ub/linux - kernel - development - best - practices - 11c1474704d6