Developing CI infrastructure for the DRM subsystem

Marcelo Mendes Spessoto Junior

Advisors
Paulo Meirelles Rodrigo Siqueira

University of São Paulo
Institute of Mathematics and Statistics

1 Introduction

Continuous Integration (CI) is a major aspect of modern software development and DevOps. By exposing code modifications to automated testing pipelines, it allows a project to be constantly modified with better consistency and safety. Therefore, it is fundamental to implement CI for free software projects with large codebases, where the contributing process needs higher coordination and many different aspects of code are being handled simultaneously. Such a project may be the Linux kernel.[2]

The Linux kernel is a very complex and large free/libre/open-source software project. It is divided in many different subsystems, each of them with their unique role.[3]

The Direct Rendering Manager (DRM) is one of the biggest subsystems of Linux. It is responsible for handling the graphical aspects of an operating system, and its codebase contains a wide variety of different GPU device drivers with their unique coding community and coding guidelines and workflow.

The Linux AMD community has an intense presence in this scenario, and a big portion of the DRM codebase is composed of AMD graphic card device drivers. Their CI, however, is still very restrictive. AMD maintainers have access to a closed CI pipeline, and submitting contributors' code to it requires some extra effort from them. There is, therefore, a need for an open and more automated CI infrastructure to be used by AMD for pre-submission testing, i.e., testing the patch before it is applied to the kernel tree.

There are some CI alternatives for the kernel context, such as DRM-CI[1] pipeline, which uses KernelCI[2] infrastructure. There is, however, an interest in a self-owned solution.

A federated CI infrastructure for the kernel at the University of São Paulo is planned for the future. This objective requires the implementation of two different aspects of a CI infrastructure: the bare-metal infrastructure where the CI pipeline will be run and software-based tools for integration of this pipeline with the code repository.

This project aims to implement the software-based integration of the pipeline with the repository. To do so, it will be necessary to implement complete software to interact with the kernel mailing lists and the CI pipeline.

2 Objectives

It is initially expected that this project will deliver software that enables the collection of recently submitted patches in a kernel mailing list, and deliver it to a CI pipeline. It then should be able to receive the evaluation results of the pipeline and be able to deliver a report back to the mailing list thread.

3 Methodology

To accomplish such objective, it will be developed a Go application that executes the following tasks:

[1]https://www.collabora.com/news-and-blog/blog/2024/02/08/drm-ci-a-gitlab-ci-pipeline-for-linux-kernel-testing/

[2]https://foundation.kernelci.org/

- Communicate with the Lore API: The Kernel Lore is a public archive containing all messages sent to kernel development mailing lists. It will be necessary to use this API to track a specific mailing list to obtain the most recent patches. The patch-hub[1] feature from kworkflow[3] can be used as a base for this feature.
- Communicate with the CI infrastructure: The application should be able to send the recent patches to the pipeline that will build, test, and deploy. It should also retrieve the testing results from a given patch. In the future, it is expected to communicate with the self-owned CI lab at IME-USP. However, since the implementation of the bare-metal infrastructure is not officially in the scope of this project, it may use external CI labs by interfacing with kernel CI pipelines such as DRM-CI.
- Send mails to Kernel Lore: the application should be able to report the CI results to the mailing list thread where the patch was sent.

4 Expected Schedule

The project is expected to be completed following this schedule:

- April: Plan the general software structure.
- May to June: Develop the lore monitoring and patch fetching features.
- July to September: Integrate the application with kernel CI pipelines.
- October: Implement a system to send reports.
- November to December: Document project. Polish application and monography.

[3]https://github.com/kworkflow/kworkflow

References

[1] David de Barros Tadokoro. *Integrating the KWorkflow system with the Lore archives.* https://linux.ime.usp.br/~davidbt/mac0499/mac0499_monograph_DavidDeBarrosTadokoro.pdf.

[2] Leonardo Leite et al. *A Survey of DevOps Concepts and Challenges.* https://dl.acm.org/doi/10.1145/3359981?cid=81413601887.

[3] Melissa Wen. *What Happens When The Bazaar Grows.* https://www.teses.usp.br/teses/disponiveis/45/45134/tde-07092021-041136/publico/dissertacao_mestrado_final_melissa_wen_10662130.pdf.