Yocto Project Long Term Support (LTS) plan proposal

Current Position

The Yocto Project is very much in favour of having some form of extended support for certain releases. The main reason this has not happened so has is due to resource constraints. We currently struggle to keep stable releases maintained for their 1-2 year lifespan, the project is therefore reluctant to committing to more work without resources to do it.

Background

Yocto project releases cadence is every six months (twice a year), as covered on the wiki https://wiki.yoctoproject.org/wiki/Releases. Every release consists of:

- Major component upgrades
    - Includes ABI/API changes
    - Include major version upgrades
    - New features
- Bug fixes reported to yocto project
- New yocto project tooling features
    - Test infrastructure changes
    - Automation changes
- New architectures added/removed

This works great for keeping a tighter integration loop with upstream.

Current Stable Releases

The project maintains stable releases for 1 year and then it moves to community support, see https://wiki.yoctoproject.org/wiki/Stable_branch_maintenance which receives no testing on AB and occasional patches for really breaking things but no regular bug fixes and security updates. No major ABI breaking patches are applied to stable releases. Occasional bug fix only version bumps for packages are accepted after review. The current stable policy says that there are acceptable and unacceptable changes:

Acceptable:

- Security and CVE fixes
- Fixes for bugs
- Fixes so codebase works with newly released distros
- Bug fix only version upgrades (especially where follows upstream policy)

Unacceptable:

- General version upgrades
- New Features

Long Term Stable (LTS) for Yocto Project

There has been rising requirement and interest amongst the project's members and its end-users for supporting a given release for longer than what a stable release is maintained. The following is a proposal for what the project could do with some thoughts on the specific decisions the project would have to make to allow this to happen. There are specific choices which would have to be made. Maintaining such an LTS will require resourcing and its likely that the people providing the resourcing will have influence over the final choices.

Term

The term is not yet determined but in principle could be anywhere between 3-5 years, maybe even longer, it ultimately depends on people being available to do the work. There are downstream projects which are relying on Yocto project releases and has a longer life cycle but are not using commercial OS Vendor solutions e.g. AGL, Microsoft Azure, RDK etc. They all are following their own cadence of deploying a given version of Yocto project release and do not join efforts since its completely driven by their own requirements. This would be an opportunity for the project to converge these projects onto a LTS release and create a large enough developer community to support a given LTS release.
Proposal: We start with a 3 year plan but this could extend if there was demand and support for maintaining it. Would need dedicated commitment of resources at the start and later a commitment to extend.

Picking an LTS

This is hard with many different viewpoints. The first decision is whether to choose an LTS in advance or afterwards, elevating an existing stable release to LTS status. There are pros and cons to both. As a point of reference, the kernel has tried both but settled on deciding in advance.
Proposal: We pick and announce an LTS in advance as this stands a better chance of people being able to align on it. This also allows the release to have focus on ensuring component choices have better long term support where possible. This implies 3.1 is the first viable candidate.

Policies within LTS

A key question is whether the LTS follows the usual stable policies. In particular, how to handle the kernel versions in the LTS is a key question and could conflict with the "no upgrades" rule usually applied to a stable series. Certainly, an LTS release will likely only maintain LTS kernels. There is a possibility that multiple LTS kernels could exist in the project LTS release. For general

recipe upgrades we'd follow stable policy which allows them in limited circumstances where upstream have a stable series or stable support model.
Whilst a "master first" policy is essential, for practicality not all stable intermediate releases may receive updates the LTS gets.
Proposal: We initially support the LTS the original release shipped with. We'd evaluate other similar vintage LTS kernels on a case by case basis depending on the status of upstream support. We need to be aware of the impact on test matrix if additional kernel versions were added and it would need to be resourced. The version of linux-libc-headers would not change to avoid user-space problems.

How often would there be an LTS

Proposal: Initially aim for an LTS every 2 years as otherwise there would be too many LTS in parallel. Ultimately it would depend on resourcing.

Components to be covered

The project components to be covered would need to match those included in our standard release process and should be clearly defined. Those components would be:

- Bitbake
- OE-Core
- Meta-yocto
- yocto-docs

(no meta-mingw or meta-gplv2)
(no vendor layers)

Who makes the final decision

Proposal: The TSC is the ultimate decision making body but it would make a decision based on community feedback, people committing resources and input from the member organisations.

LTS Maintainership

A LTS Maintainer is selected in the same manner as the Stable releases, i.e. the repo owner has the call. There could be more than one person who has ownership of the LTS branch but one may be identified as the point person. The Point person would be the one who handles bugzilla ownership/queries, build, QA and backporting concerns.

The Yocto TSC would retain the QA test result review and release go/nogo decision for any releases.

The LTS maintainer will be responsible in starting and monitoring builds. The Maintainer may have assistance from the community in resolving new issues identified during build and or the QA run.

Backport reviews will be sent to the mailing list for community review.
A merge request will be sent to the appropriate repo owner once all issues found during the review have been addressed.

Infrastructure Needed

Whilst not immediately obvious and whilst well suited to testing current development, the current autobuilder is not suited to building, maintaining and testing an LTS release. In particular the autobuilder workers are multiple different distros (to get wide test coverage) running "bare-metal" for performance. For security reasons we only have workers which are in current support and have upgrade feeds available. We already struggle with the stable branches in this area.

In order to support an LTS release for the project, we'd propose that new autobuilder software infrastructure needs to be developed to support it. This could run on the same autobuilder hardware but we'd propose that for LTS, only one host distro be used for testing, probably an Ubuntu LTS because its easily available and has a long lifetime. This would most likely be in the form of a container based worker and the specific distro used would be used throughout the lifetime of the project LTS release. This would imply a worker/container combination per LTS release.

Building such software infrastructure is definitely possible but also non-trivial and as such, the work in setting it up needs to be included in any LTS plan.

An alternative could be to have a larger number of nodes running the chosen LTS distro and limiting the LTS builds to those workers. This could have implications for the build/testing time of the release. It could also have an impact on the availability of specialist processing workers such as the native ARM one. Performance testing is another area which would have to be carefully considered as the build time performance testing workers may not run the supported LTS distro.

For simplicity, it is also proposed that only automated testing be used for testing the LTS and that any current manual QA is not performed. The main reason for this is to streamline and simplify the testing and release process to allow regular and frequent updates to the LTS release without dependencies on external factors. Anyone can obviously perform their own testing of the LTS releases in addition to this core automated testing.

Proposal:

- Follow the same testing process as the original release
- Only run virtualized tests
- Only support one host distro, an Ubuntu LTS as it has right lifespan

- Start by aiming to share the infrastructure meaning multiple LTS workers or LTS worker containers depending on funding and implementation
- To share infrastructure this implies one autobuilder controller covering both potentially complicating configuration changes for master (likely manageable)

Resource Requirements/Summary

The infrastructure requirements are easier to quantify and would likely consist of 2-3 additional additional worker machines over time for the above proposal as the minimum cost.

The human resource element to track, test (using automation) and merge patches is harder to quantify but at a rough guess, is probably 50% of a person's time for the lifespan of the LTS.

This assumes that patches for various issues are forthcoming from others as its not realistic to expect one maintainer to handle creation of the various patches needed. The quality of the LTS will be directly related to the number of people working on the patches and them having the time to be able to ensure the patches are of the needed quality.

Other Considerations

- Migration from one LTS to another
- Package compatibility from one LTS to another
- Should support subset of machines? Architectures?
- Prior Art Ubuntu: normal releases every six months (April and October), LTS every two years (every other April). Details and cadence chart at https://ubuntu.com/about/release-cycle.
- How we identify LTS releases (os-release, tags, wiki), pros and cons to changing
- If dynamic components such as go or rust are in core we may need a way to allow them to change at a higher rate of change due to the nature of the languages. Likely this can be done through an additional layer alongside the LTS to have the newer versions which users can collaborate on together. LTS "mixin" layers?