Sprint 1 Planning Document

Team YFFS

Team 22
Brian VerVaet, Max Molnar, Sam Spencer, Wyatt Dahlenburg, Zach Kent

Table of Contents

Sprint Overview

A strong foundation is important to the success of any large project. With that in mind our team is using sprint one to focus on the essential functionality of the YFFS file system. This means modifying and documenting the all of the steps that currently exist in the basic filesystem we are using. In order to ensure we are prepared to implement the advanced features we hope to create, we must guarantee that YFFS is working as expected. To do this we will be splitting the project into many smaller components and developing tests for each. In addition to testing existing functionality we will be improving implementation so that all read and write operations will be thread safe. With these steps completed our team will be well prepared to tackle the remainder of the backlog over the next two sprints.

Scrum Master: Zachary Kent

Meeting Schedule: Tuesdays, Thursdays, and Sundays at 3:00pm

Risks/Challenges: While all members of our team are well versed in the C programming language, this project will stretch our programming skills and force our team to develop in a way we never have before. Learning new skills and topics will require team members to contribute time and effort independently before they are prepared to begin working on many of the sprint tasks. This can cause a problem if one or more team members is not able to complete their task either because they did not prepare or they cannot find the necessary resources.

Current Sprint Detail

User Story #1

- As a user, I would like to be able to create a YFFS filesystem.

#	Task Description	Estimated Time	Owner
1	Research file system image creation.	2 Hrs.	Team
2	Document current creation functionality.	2 Hrs.	Zachary
3	Test the current creation framework.	3 Hrs.	Zachary
4	Reorganize all code and create a Makefile that allows for clean creation of more tools.	1 Hr.	Zachary
5	Create an independent tool to create the files (yffs-create).	3 Hrs.	Zachary
6	Debug and test yffs-create.	3 Hrs.	Zachary

Acceptance Criteria:

- YFFS tools should be easy to install, with the option to uninstall.
- If a user has directory write permissions, the user can create a filesystem there.
- A user can create a filesystem with any amount the of available memory.
- Only the user who created the filesystem will have read or write permissions unless explicitly adjusted.

User Story #2

- As a user, I would like to be able to write and add files to YFFS.

#	Task Description	Estimated Time	Owner
1	Rewrite write functions so that they are atomic	4 Hrs.	Max
2	Implement write permissions in file data structure and algorithm	4 Hrs.	Sam
3	Create an independent tool to write to files in YFFS (yffs-write)	1 Hr.	Max
4	Debug and test yffs-write	2 Hrs.	Max
5	Create an independent tool to add files to YFFS (yffs-touch)	1 Hrs.	Sam
6	Debug and test yffs-touch	2 Hrs.	Sam

Acceptance Criteria:

- If a user has write permissions, the user can write to any files in the file system.
- If YFFS experiences power loss during a write procedure, memory will not be corrupted.
- yffs-touch and yffs-write command should be easy to use and work similar to the unix commands.

User Story #3

- As a user, I would like to be able read files stored in YFFS.

#	Task Description	Estimated Time	Owner
1	Research atomic functionality	1 Hr.	Max
2	Rewrite read functions so that they are atomic	3 Hrs.	Max
3	Implement read permissions in file data structure and algorithm	3 Hrs.	Sam
4	Create an independent tool to list files (yffs-cat)	3 Hrs.	Sam
5	Test and debug yffs-cat	1 Hr.	Sam

Acceptance Criteria:

- If a user has read permissions, the user can read any files in the file system.
- If YFFS experiences power loss during a read procedure, memory will not be corrupted.
- yffs-cat should be easy to use and work similar to the unix ‘cat’ command.

User Story #4

- As a user, I would like to be able to remove files from YFFS.

#	Task Description	Estimated Time	Owner
1	Research atomic functionality / thread safety	1 Hr.	Brian
2	Rewrite remove functions to be atomic	3 Hrs.	Brian
3	Create an independent tool to remove files (yffs-rm)	3 Hrs.	Brian
4	Implement editing permissions into algorithm	2 Hrs.	Brian
5	Debug and test yffs-rm	1 Hr.	Brian

Acceptance Criteria:

- If a user has permissions, the user can remove any file in the file system.
- If the file system experiences power loss during a remove procedure, the system will not be corrupted.
- yffs-rm will not remove any file currently in use by another tool.
- yffs-rm should be easy to use and work similar to the unix ‘rm’ command.

User Story #5

- As a user, I would like to be able to list all files stored in YFFS.

#	Task Description	Estimated Time	Owner
1	Create an independent tool to list files (yffs-ls)	3 Hrs.	Sam
2	Add permissions to listing algorithm	2 Hrs.	Brian
3	Prepare for the obfuscation interface	2 Hrs.	Brian
4	Add functionality to export file list	3 Hrs.	Brian
5	Debug and test yffs-ls	1 Hr.	Sam

Acceptance Criteria:

- If a user has permissions, the user can list all files in the file system.
- yffs-ls should always have the most up-to date file list.
- List export location may be defined by the user.
- yffs-ls should be easy to use and work similar to the unix ‘ls’ command.

User Story #6

- As a user, I would like to have dependable data in the event of power failure.

#	Task Description	Estimated Time	Owner
1	Research normal power loss situations	3 Hrs.	Team
2	Setup raspberry pi with YFFS	2 Hrs.	Wyatt Dahlenburg
3	Write program to simulate power loss	4 Hrs.	Wyatt Dahlenburg
4	Create power spikes and loss events in a program and test the fault-tolerant file system	3 Hrs.	Wyatt Dahlenburg

Acceptance Criteria:

- YFFS should be fully functioning on a raspberry pi.
- Our testing solution should be able to simulate both power spikes and total power loss.
- YFFS should not be corrupted even if we force a power event during in the middle of a read or write operation.

Remaining Backlog

Functional Requirements

- As a user I would like to be able to,
  - create directories for files.
  - use this file system on embedded devices.
  - have a complete documentation of the filesystem tools.

- As a developer, I would like a(n)
  - simple set of commands for the filesystem.
  - API to assist with the inclusion of encryption plug-ins.

- As an admin, I would like
  - to maintain permissions of files.
  - to define encryption operations on the filesystem.
  - to define a hash for all filenames to be obfuscated.

Non-Functional Requirements

- We must be able to implement this on many different devices.
- The encryption interface should work quickly.
- The system will be mountable.
- The system will run efficiently on embedded devices.
- Documentation will be thorough and robust.
- The filesystem and encryption interface should be easy to use.