Spruce technical documentation

Contents

General

The Spruce system is designed for quality verification of Ext4, JFS, XFS and BtrFS file system drivers. It is developed in C++ language using XML-based template engine for test generation. To be able to run the tests for a particular file system one should have the corresponding tools installed. For example to be able to run tests on JFS partition, the jfsutils package must be installed. It consists of two subsystems:

- configuration, build, execution and log analysis subsystem,
- test subsystem.

Each of these subsystems consists of one or more modules (see details in next section). The configuration mechanism uses the CMake platform. In that phase several system-based values and other configurable things are set which are later used in the build process. Build system uses GNU Make tool. After the build is complete several binary files generated which need to be installed (along with some other files). For details on system configuration, build and execution see the “Spruce user Manual” paper. After the execution a dashboard HTML file is generated and opened in browser (depends on the values in config file). From that file all the concrete log files can be accessed. Detailed explanation of log files see in section “Log

analysis”.

The template engine

The Spruce system is developed on C++ language using all the advantages of the object-oriented programming. But still even classes and objects are not enough in test generation processes. For purposes one would usually need some templates mechanism to be used. The Spruce system uses its own template engine. The engine is based on XML. IN terms of the engine a module is an XML file where its test sets are defined. Each test set is also an XML file with the listing of its tests. A test is just an element in the test set XML file represented by the <Test> tag. Tests can have some attributes like name and fault simulation ready flag, a description, header and footer sections and the code to be executed. On build stage the engine itself is build to process the XMl files. After transformation each module consists of a module.cpp file and a set of .hpp files one per test set.

Modules

The Spruce system operates based on modules. All the aspects of the system execution, starting with the user interaction and ending with the log generation and visualization are realized through modules. Module is a component of the system which performs a single task (e.g. does testing of a certain file system driver). There are 3 types of modules: module manager, meta-modules and worker modules.

Main

The main module is the module manager. It is responsible for the following tasks:

- user interaction,
- configuration file parsing,
- partition preparation,
- worker and meta-module execution,
- log generation and visualization.

The *main* module executable is called “spruce”. It is installed under ${CMAKE_INSTALL_PREFIX}/bin folder. To execute it a configuration file must be prepared and passed to the main module executable via -c <file_path> arguments. The supported keys and their possible and default values are explained in the “Spruce user manual” paper. If the configuration file is provided main module parses it, finds out the provided values, adds default values for the missed keys. Among the keys in the config file the file systems and modules must be specified. Currently there are 4 supported file systems: JFS, Ext4FS, XFS and BtrFS. If all the provided file system names and module names are correct then main module prepares the partition (also provided in the config file) for tests execution. Thus it creates the corresponding file system there and mounts the partition. In fact mounting is done for all the supported mount

options to get higher level of testing. This operation is performed for all the mentioned file systems. Then for each file system all the mentioned modules are executed. The worker modules store their output in XML format. After each worker module exits the main module transforms the generated log file into HTML.

At the end if there was provided a browser name in the configuration file then the main module executes the browser opening the log files of the modules. Otherwise the system considered to be executed in “batch mode”. Thus it must just return the status code. The status code for the whole system is formed of the status codes returned by all the executed modules. So if there is any test that failed or was not resolved or ended with another unsuccessful status then the status of the whole Spruce execution is also non-zero.

Syscall

One of largest moduls in the Spruce system is the *syscall* module. It is responsible for the syscalls verification. This means that the syscall module checks if the file system driver behavior corresponds the POSIX documentation. After experiments several system calls were found which were not covered by the system. The list includes the system calls ftruncate, quotactl and others. It includes tests for many FS-oriented system calls, such as open, read, write, fcntl, etc. As bases are taken the POSIX man pages. Almost all of them have the sections about the system call normal functionality and error cases. According to it all the syscall tests are divided into two groups: normal functional tests and error tests. The error tests check for all the possible error codes returned by the corresponding system in error cases. There is a set of error codes and scenarios that can be found in almost every system call man page. Examples: ENOENT, EBADF, ENAMETOOLONG. That’s why such tests are so said standardized.

FS-Specific modules

FS-specific modules are responsible for testing of the functionality which is specific for the current file system. Not all the specific things are described in the POSIX man pages. usually their explanations can be found in the wiki pages of the file system project or in the commit logs or even in the comments in the source code. It makes really hard to standardize the FS-specific verification. That’s why there are modules for each supported file system drivers.

Ext4

The *Ext4* module is responsible for the Ext4FS-specific testing. Ext4 has several functionalities which are not covered by the standard system calls (more precisely with standard argument values of system calls). There is a special system call, *ioctl* - input/output control. Mainly all the FS-specific things are realized through this system call. The Ext4 module contains tests for several operations supported by the ioctl system call, such as:

- set and get flags
- set and get version
- move extents (used in defragmentation)
- resize
- perform non-permitted actions

- perform unsupported operation
- etc.

Some of these operations can be really checked but some of them are just activated (shallow level tests).

JFS

At the moment the JFS file system almost have no specific functionalities. So the *JFS* module is just a placeholder for future developments.

XFS

The XFS file system is one of the largest file systems supported by Linux. It has a really large set of specific operations, most of which are still checked only in shallow level. The set of XFS-specific tests include tests concerning:

- geometry
- space manipulation
- xflags
- xattributes
- etc.

BtrFS

BtrFS is simply the largest file system which is being verified by the Spruce system. It has the largest set of specific functionalities. Mostly they are available through the same ioctl system call. The set of tested functionalities includes test concerning:

- snapshots
- subvolumes
- defragmentation
- resize
- device information
- etc.

At the moment most of these tests are on shallow level.

Fault Simulation

One of the goals of the Spruce system is to ensure that the file system driver does not crash in case of rare execution paths. Such paths mainly include error checking and handling. To be able to that some fault simulation mechanism is needed. Spruce uses the KEDR framework (http://code.google.com/p/kedr) for fault simulation (and not only for that). KEDR allows to define really well-tuned scenarios for a wide set of kernel functions to fail. For example it can be set to fail all the memory allocation functions when they are called from a certain process and under certain conditions (like each 5-th function call).

Spruce *fault-sim* module integrates the KEDR framework to Spruce to be able to check the driver behaviour in faulty and rare cases. This process is really dangerous because FS drivers may react in different ways to such situations and usually after fault-sim module is

executed once the whole system needs to be restarted.

Memory leak checker

Another important feature of the Spruce system memory leak detection. For this purpose also the KEDR framework is used. It can report about possible leaks and use-after-free situations within a certain kernel module. These reports are handled by the *leak-check* modules and reported to the user in the same way as all the other error found during the Spruce execution.

Log analysis

During the Spruce execution log files are created for each module. At the beginning they are XML files. If the browser name is set in the configuration file then the main module transforms the XML files to HTML and open them in the browser. So for each file system and for each modules executed on that file system we'll have one log file.

After the system is executed another log file is generated which show some aggregate information: for each mount option and for each module it shows summaries of passed, failed and unresolved tests. From this log file all other concrete log files can be accessed.

All the log files have the same format: aggregate information and details. In the first section the counts are provided by test statuses. There are the following statuses supported by the Spruce system:

- Success - the test has checked the corresponding functionality and it succeeded.
- Shallow - the test has just enabled the corresponding functionality and it succeeded.
- Fail - the corresponding functionality did not pass the test.
- Unresolved - it was not possible to check the corresponding functionality.
- Fatal - some error has raised which does not allow to continue the module execution.
- Signalled - the process has received a signal.
- Unsupported - the corresponding functionality is not supported by the system.
- Skipped - the test was skipped according to the configuration file values.

Expanding any of the status lines one can see the names of certain tests that have ended with that status along with it's output.