Using Sigmund

Un article de WikIGS.

Sommaire

Using the Sigmund Test Suite

Sigmund is an export about regresssions ;-) It has been designed to detect regression in filesystems in order to help locating them and curing them.

Sigmund is modular and fully written in bash to ease extensibility.

Configuring sigmund

In order to run Sigmund, you'll need a few programs. They are well known products and no part of sigmund, so I don not integrate them in the repository. There may be license conflict as well.

Getting the prerequiste

The prerequisite are:

- the cthon04 test suite from Oracle
- the pynfs test suite

Getting cthon04

```
# wget http://hub.opensolaris.org/bin/download/Community+Group+nfs/tests/nfstests.tar.gz
--2013-03-19 10:07:18--  http://hub.opensolaris.org/bin/download/Community+Group+nfs/tests/nfstests.tar.gz
(...)
Length: 90309 (88K) [application/x-gzip]
Saving to: `nfstests.tar.gz'
(...)
```

Untar the file in /opt (or any other location of your choice). You'll get a tree whose root is named ***cthon04*** Then edit the c***thon04/tests.init*** file and make it look this:

```
MNTOPTIONS="rw,hard,intr"
MNTPOINT="/mnt"

MNTOPTIONS="rw,intr"
MOUNTCMD='echo'
UMOUNTCMD='echo'

DASHN=-n
BLC=

PATH=/bin:/usr/bin:/usr/ucb:/usr/ccs/bin:/sbin:/usr/sbin:.

MOUNT=/usr/sbin/mount
UMOUNT=/usr/sbin/umount

CC=gcc
CFLAGS=-g
LIBS=
LOCKTESTS=`echo tlocklfs tlock64`
```

And finally, run make

```
gcc -g -o domount domount.c
domount.c: In function 'main':
domount.c:35:2: warning: incompatible implicit declaration of built-in function 'exit' [enabled by default]
chown root domount && chmod u+s domount
chown: changing ownership of `domount': Operation not permitted
make: [domount] Error 1 (ignored)
gcc -g -o getopt getopt.c
getopt.c: In function 'main':
getopt.c:109:3: warning: incompatible implicit declaration of built-in function 'exit' [enabled by default]
getopt.c:115:4: warning: incompatible implicit declaration of built-in function 'exit' [enabled by default]
getopt.c:141:2: warning: incompatible implicit declaration of built-in function 'exit' [enabled by default]
cd basic; make
make[1]: Entering directory `/tmp/cthon04/basic'
gcc -g   -c -o subr.o subr.c
gcc -g -o test1 test1.c subr.o
gcc -g -o test2 test2.c subr.o
gcc -g -o test3 test3.c subr.o
gcc -g -o test4 test4.c subr.o
gcc -g -o test5 test5.c subr.o
gcc -g -o test6 test6.c subr.o
gcc -g -o test7 test7.c subr.o
gcc -g -o test8 test8.c subr.o
gcc -g -o test9 test9.c subr.o
gcc -g -o test4a test4a.c subr.o
gcc -g -o test5a test5a.c subr.o
gcc -g -o test5b test5b.c subr.o
gcc -g -o test7a test7a.c subr.o
gcc -g -o test7b test7b.c subr.o
if test ! -x runtests; then chmod a+x runtests; fi
make[1]: Leaving directory `/tmp/cthon04/basic'
cd general; make
make[1]: Entering directory `/tmp/cthon04/general'
rm -f large1.c
cp large.c large1.c
rm -f large2.c
cp large.c large2.c
rm -f large3.c
cp large.c large3.c
if test ! -x runtests; then chmod a+x runtests; fi
make[1]: Leaving directory `/tmp/cthon04/general'
cd special; make
make[1]: Entering directory `/tmp/cthon04/special'
gcc -g -o op_unlk op_unlk.c
gcc -g -o op_ren op_ren.c
gcc -g -o op_chmod op_chmod.c
gcc -g -o dupreq dupreq.c
gcc -g -o excltest excltest.c
gcc -g -o negseek negseek.c
gcc -g -o rename rename.c
gcc -g -o holey holey.c
gcc -g -o truncate truncate.c
gcc -g -o nfsidem nfsidem.c
gcc -g -o nstat nstat.c ../basic/subr.o
gcc -g -o stat stat.c ../basic/subr.o
gcc -g -o stat2 stat2.c ../basic/subr.o
gcc -g -o touchn touchn.c
gcc -g -o fstat fstat.c
gcc -g -o rewind rewind.c
gcc -g -o telldir telldir.c ../basic/subr.o
gcc -g -o bigfile bigfile.c ../basic/subr.o
gcc -g -o bigfile2 bigfile2.c ../basic/subr.o
gcc -g -o freesp freesp.c ../basic/subr.o
if test ! -x runtests; then chmod a+x runtests; fi
make[1]: Leaving directory `/tmp/cthon04/special'
cd lock; make
make[1]: Entering directory `/tmp/cthon04/lock'
make[2]: Entering directory `/tmp/cthon04/lock'
gcc -g -DLF_SUMMIT -o tlocklfs tlock.c -lm
gcc -g -DLF_SUMMIT -DLARGE_LOCKS -o tlock64 tlock.c -lm
make[2]: Leaving directory `/tmp/cthon04/lock'
if test ! -x runtests; then chmod a+x runtests; fi
make[1]: Leaving directory `/tmp/cthon04/lock'
if test ! -x runtests; then chmod a+x runtests; fi
```

Getting pynfs

The quickest and best way is to git clone the repository

```
# git clone git://linux-nfs.org/~bfields/pynfs.git
Cloning into 'pynfs'...
remote: Counting objects: 2327, done.
remote: Compressing objects: 100% (1025/1025), done.
remote: Total 2327 (delta 1679), reused 1807 (delta 1279)
Receiving objects: 100% (2327/2327), 924.63 KiB | 464 KiB/s, done.
```

Setting Up Sigmund

There are 2 steps

- Configuring run_test.tc
- building the tests

Write the configuration file

In sigmund's root directory, edit the run_test.rc file It will look like this:

```
##### Root of test  #####
TEST_DIR=/mnt/sigmund
BUILD_TEST_DIR=/tmp/sigmund

##### Variables to be used by module allfs #####
# Path to cthon04 test's suite
CTHON04_DIR=/opt/cthon04
GIT_CLONE_URL=/opt/GANESHA/.git

# Non-root user that will run part of the test
TEST_USER=root
```

Important variables are

- **TEST_DIR**: the directory, inside the mount point to be tested, were Sigmund will be run. Make sure that **TEST_USER**can read and write in it
- **BUILD_TEST_DIR**: a temporary directory where a few C programs will be generated. Make sure that **TEST_USER**can read and write in it
- **CTHON04_DIR**: the directory where ctho04 was installed. **TEST_USER** must be abe to read/execute in it.
- **GIT_CLONE_URL** : a git repository that **TEST_USER**can read (even in a read-only way). I recommend a local repo for generating a bigger stress to the filesystem

Build the tests

The procedure is quite simple

```
# ./build_test.sh
test_mmap_read compiled and moved to  /tmp/sigmund
test_mmap_write compiled and moved to  /tmp/sigmund
```

Running Sigmund

The default way of running sigmund will starts all of the tests. Please not that, even if TEST_USER is not root, the tests are to be run as root (there will be later calls to "su"). You can do it in three way:

- running it in verbose mode : use **./run_test.sh**
- running it in verbose mode with extra JUnit report. This is designed for integration in jenkins : **./run_test.sh -j**
- running it in quiet mode : use **./run_test.sh -q**

When run in quiet mode, ouput will look like this:

```
# ./run_test.sh -q
test1 :  ALLFS: copy file with 444 mode      [  OK  ]
test2 :  ALLFS: rm -rf of wide namespace     [  OK  ]
test3b : ALLFS: cthon04's basic tests        [  OK  ]
test3g : ALLFS: cthon04's general tests      [  OK  ]
test3s : ALLFS: cthon04's special tests      [  OK  ]
test3l : ALLFS: cthon04's lock tests         [  OK  ]
test4g : ALLFS: git clone a local repository [  OK  ]
test4s : ALLFS: Tar calls utimes on symlink  [  OK  ]
test4r : ALLFS: Use mmap() to read a file    [  OK  ]
test4w : ALLFS: Use mmap() to write a file   [  OK  ]
All tests passed (10 successful, 0 skipped)
```

You can restrict t a subset of the tests by setting the **ONLY** variable.

```
# ONLY=4 ./run_test.sh -q
test4g : ALLFS: git clone a local repository [  OK  ]
test4s : ALLFS: Tar calls utimes on symlink  [  OK  ]
test4r : ALLFS: Use mmap() to read a file    [  OK  ]
test4w : ALLFS: Use mmap() to write a file   [  OK  ]
All tests passed (4 successful, 0 skipped)
```

or (a more sophiticated example)

```
# ONLY=1,3,4s ./run_test.sh -q
test1 :  ALLFS: copy file with 444 mode      [  OK  ]
test3b : ALLFS: cthon04's basic tests        [  OK  ]
test3g : ALLFS: cthon04's general tests      [  OK  ]
test3s : ALLFS: cthon04's special tests      [  OK  ]
test3l : ALLFS: cthon04's lock tests         [  OK  ]
test4s : ALLFS: Tar calls utimes on symlink  [  OK  ]
All tests passed (6 successful, 0 skipped)
```