Fault injection capabilities infrastructure

See also drivers/md/md-faulty.c and "every_nth" module option for scsi_debug.

Available fault injection capabilities

- failslab

  injects slab allocation failures. (kmalloc(), kmem_cache_alloc(), ...)

- fail_page_alloc

  injects page allocation failures. (alloc_pages(), get_free_pages(), ...)

- fail_usercopy

  injects failures in user memory access functions. (copy_from_user(), get_user(), ...)

- fail_futex

  injects futex deadlock and uaddr fault errors.

- fail_sunrpc

  injects kernel RPC client and server failures.

- fail_make_request

  injects disk IO errors on devices permitted by setting /sys/block/<device>/make-it-fail or /sys/block/<device>/<partition>/make-it-fail. (submit_bio_noacct())

- fail_mmc_request

  injects MMC data errors on devices permitted by setting debugfs entries under /sys/kernel/debug/mmc0/fail_mmc_request

- fail_function

  injects error return on specific functions, which are marked by ALLOW_ERROR_INJECTION() macro, by setting debugfs entries under /sys/kernel/debug/fail_function. No boot option supported.

- NVMe fault injection

  inject NVMe status code and retry flag on devices permitted by setting debugfs entries under /sys/kernel/debug/nvme*/fault_inject. The default status code is NVME_SC_INVALID_OPCODE with no retry. The status code and retry flag can be set via the debugfs.

Configure fault-injection capabilities behavior

debugfs entries

fault-inject-debugfs kernel module provides some debugfs entries for runtime configuration of fault-injection capabilities.

- /sys/kernel/debug/fail*/probability:

  likelihood of failure injection, in percent.

  Format: <percent>

  Note that one-failure-per-hundred is a very high error rate for some testcases. Consider setting probability=100 and configure /sys/kernel/debug/fail*/interval for such testcases.

- /sys/kernel/debug/fail*/interval:

  specifies the interval between failures, for calls to should_fail() that pass all the other tests.

  Note that if you enable this, by setting interval>1, you will probably want to set probability=100.

- /sys/kernel/debug/fail*/times:

  specifies how many times failures may happen at most. A value of -1 means "no limit". Note, though, that this file only accepts unsigned values. So, if you want to specify -1, you better use 'printf' instead of 'echo', e.g.: $ printf %#x -1 > times

- /sys/kernel/debug/fail*/space:

  specifies an initial resource "budget", decremented by "size" on each call to should_fail(,size). Failure injection is suppressed until "space" reaches zero.

- /sys/kernel/debug/fail*/verbose

  Format: { 0 | 1 | 2 }

  specifies the verbosity of the messages when failure is injected. '0' means no messages; '1' will print only a single log line per failure; '2' will print a call trace too -- useful to debug the problems revealed by fault injection.

- /sys/kernel/debug/fail*/task-filter:

  Format: { 'Y' | 'N' }

  A value of 'N' disables filtering by process (default). Any positive value limits failures to only processes indicated by /proc/<pid>/make-it-fail==1.

- /sys/kernel/debug/fail*/require-start, /sys/kernel/debug/fail*/require-end, /sys/kernel/debug/fail*/reject-start, /sys/kernel/debug/fail*/reject-end:

  specifies the range of virtual addresses tested during stacktrace walking. Failure is injected only if some caller in the walked stacktrace lies within the required range, and none lies within the rejected range. Default required range is [0,ULONG_MAX) (whole of virtual address space). Default rejected range is [0,0).

- /sys/kernel/debug/fail*/stacktrace-depth:

  specifies the maximum stacktrace depth walked during search for a caller within [require-start,require-end) OR [reject-start,reject-end).

- /sys/kernel/debug/fail_page_alloc/ignore-gfp-highmem:

  Format: { 'Y' | 'N' }

  default is 'N', setting it to 'Y' won't inject failures into highmem/user allocations.

- /sys/kernel/debug/failslab/ignore-gfp-wait:
- /sys/kernel/debug/fail_page_alloc/ignore-gfp-wait:

  Format: { 'Y' | 'N' }

  default is 'N', setting it to 'Y' will inject failures only into non-sleep allocations (GFP_ATOMIC allocations).

- /sys/kernel/debug/fail_page_alloc/min-order:

  specifies the minimum page allocation order to be injected failures.

- /sys/kernel/debug/fail_futex/ignore-private:

  Format: { 'Y' | 'N' }

  default is 'N', setting it to 'Y' will disable failure injections when dealing with private (address space) futexes.

- /sys/kernel/debug/fail_sunrpc/ignore-client-disconnect:

  Format: { 'Y' | 'N' }

  default is 'N', setting it to 'Y' will disable disconnect injection on the RPC client.

- /sys/kernel/debug/fail_sunrpc/ignore-server-disconnect:

  Format: { 'Y' | 'N' }

  default is 'N', setting it to 'Y' will disable disconnect injection on the RPC server.

- /sys/kernel/debug/fail_function/inject:

  Format: { 'function-name' | '!function-name' | '' }

  specifies the target function of error injection by name. If the function name leads '!' prefix, given function is removed from injection list. If nothing specified ('') injection list is cleared.

- /sys/kernel/debug/fail_function/injectable:

  (read only) shows error injectable functions and what type of error values can be specified. The error type will be one of below; - NULL: retval must be 0. - ERRNO: retval must be -1 to -MAX_ERRNO (-4096). - ERR_NULL: retval must be 0 or -1 to -MAX_ERRNO (-4096).

- /sys/kernel/debug/fail_function/<function-name>/retval:

  specifies the "error" return value to inject to the given function. This will be created when the user specifies a new injection entry. Note that this file only accepts unsigned values. So, if you want to use a negative errno, you better use 'printf' instead of 'echo', e.g.: $ printf %#x -12 > retval

Boot option

In order to inject faults while debugfs is not available (early boot time), use the boot option:

```
failslab=
fail_page_alloc=
fail_usercopy=
fail_make_request=
fail_futex=
mmc_core.fail_request=<interval>,<probability>,<space>,<times>
```

proc entries

- /proc/<pid>/fail-nth, /proc/self/task/<tid>/fail-nth:

  Write to this file of integer N makes N-th call in the task fail. Read from this file returns a integer value. A value of '0' indicates that the fault setup with a previous write to this file was injected. A positive integer N indicates that the fault wasn't yet injected. Note that this file enables all types of faults (slab, futex, etc). This setting takes precedence over all other generic debugfs settings like probability, interval, times, etc. But per-capability settings (e.g. fail_futex/ignore-private) take precedence over it.

  This feature is intended for systematic testing of faults in a single system call. See an example below.

How to add new fault injection capability

- #include <linux/fault-inject.h>
- define the fault attributes

  DECLARE_FAULT_ATTR(name);

  Please see the definition of struct fault_attr in fault-inject.h for details.
- provide a way to configure fault attributes
- boot option

  If you need to enable the fault injection capability from boot time, you can provide boot option to configure it. There is a helper function for it:

  setup_fault_attr(attr, str);

- debugfs entries

  failslab, fail_page_alloc, fail_usercopy, and fail_make_request use this way. Helper functions:

  fault_create_debugfs_attr(name, parent, attr);

- module parameters

  If the scope of the fault injection capability is limited to a single kernel module, it is better to provide module parameters to configure the fault attributes.
- add a hook to insert failures

  Upon should_fail() returning true, client code should inject a failure:

  should_fail(attr, size);

Application Examples

- Inject slab allocation failures into module init/exit code:

```
#!/bin/bash

FAILTYPE=failslab
echo Y > /sys/kernel/debug/$FAILTYPE/task-filter
echo 10 > /sys/kernel/debug/$FAILTYPE/probability
echo 100 > /sys/kernel/debug/$FAILTYPE/interval
printf %#x -1 > /sys/kernel/debug/$FAILTYPE/times
echo 0 > /sys/kernel/debug/$FAILTYPE/space
```

```
echo 2 > /sys/kernel/debug/$FAILTYPE/verbose
echo 1 > /sys/kernel/debug/$FAILTYPE/ignore-gfp-wait

faulty_system()
{
    bash -c "echo 1 > /proc/self/make-it-fail && exec $*"
}

if [ $# -eq 0 ]
then
    echo "Usage: $0 modulename [ modulename ... ]"
    exit 1
fi

for m in $*
do
    echo inserting $m...
    faulty_system modprobe $m

    echo removing $m...
    faulty_system modprobe -r $m
done
```

- Inject page allocation failures only for a specific module:

```
#!/bin/bash

FAILTYPE=fail_page_alloc
module=$1

if [ -z $module ]
then
    echo "Usage: $0 <modulename>"
    exit 1
fi

modprobe $module

if [ ! -d /sys/module/$module/sections ]
then
    echo Module $module is not loaded
    exit 1
fi

cat /sys/module/$module/sections/.text > /sys/kernel/debug/$FAILTYPE/require-start
cat /sys/module/$module/sections/.data > /sys/kernel/debug/$FAILTYPE/require-end

echo N > /sys/kernel/debug/$FAILTYPE/task-filter
echo 10 > /sys/kernel/debug/$FAILTYPE/probability
echo 100 > /sys/kernel/debug/$FAILTYPE/interval
printf %#x -1 > /sys/kernel/debug/$FAILTYPE/times
echo 0 > /sys/kernel/debug/$FAILTYPE/space
echo 2 > /sys/kernel/debug/$FAILTYPE/verbose
echo 1 > /sys/kernel/debug/$FAILTYPE/ignore-gfp-wait
echo 1 > /sys/kernel/debug/$FAILTYPE/ignore-gfp-highmem
echo 10 > /sys/kernel/debug/$FAILTYPE/stacktrace-depth

trap "echo 0 > /sys/kernel/debug/$FAILTYPE/probability" SIGINT SIGTERM EXIT

echo "Injecting errors into the module $module... (interrupt to stop)"
sleep 1000000
```

- Inject open_ctree error while btrfs mount:

```
#!/bin/bash

rm -f testfile.img
dd if=/dev/zero of=testfile.img bs=1M seek=1000 count=1
DEVICE=$(losetup --show -f testfile.img)
mkfs.btrfs -f $DEVICE
mkdir -p tmpmnt

FAILTYPE=fail_function
FAILFUNC=open_ctree
echo $FAILFUNC > /sys/kernel/debug/$FAILTYPE/inject
printf %#x -12 > /sys/kernel/debug/$FAILTYPE/$FAILFUNC/retval
echo N > /sys/kernel/debug/$FAILTYPE/task-filter
echo 100 > /sys/kernel/debug/$FAILTYPE/probability
echo 0 > /sys/kernel/debug/$FAILTYPE/interval
```

```
printf %#x -1 > /sys/kernel/debug/$FAILTYPE/times
echo 0 > /sys/kernel/debug/$FAILTYPE/space
echo 1 > /sys/kernel/debug/$FAILTYPE/verbose

mount -t btrfs $DEVICE tmpmnt
if [ $? -ne 0 ]
then
    echo "SUCCESS!"
else
    echo "FAILED!"
    umount tmpmnt
fi

echo > /sys/kernel/debug/$FAILTYPE/inject

rmdir tmpmnt
losetup -d $DEVICE
rm testfile.img
```

Tool to run command with failslab or fail_page_alloc

In order to make it easier to accomplish the tasks mentioned above, we can use tools/testing/fault-injection/failcmd.sh. Please run a command "./tools/testing/fault-injection/failcmd.sh --help" for more information and see the following examples.

Examples:

Run a command "make -C tools/testing/selftests/ run_tests" with injecting slab allocation failure:

```
# ./tools/testing/fault-injection/failcmd.sh \
        -- make -C tools/testing/selftests/ run_tests
```

Same as above except to specify 100 times failures at most instead of one time at most by default:

```
# ./tools/testing/fault-injection/failcmd.sh --times=100 \
        -- make -C tools/testing/selftests/ run_tests
```

Same as above except to inject page allocation failure instead of slab allocation failure:

```
# env FAILCMD_TYPE=fail_page_alloc \
        ./tools/testing/fault-injection/failcmd.sh --times=100 \
        -- make -C tools/testing/selftests/ run_tests
```

Systematic faults using fail-nth

The following code systematically faults 0-th, 1-st, 2-nd and so on capabilities in the socketpair() system call:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/syscall.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>

int main()
{
      int i, err, res, fail_nth, fds[2];
      char buf[128];

      system("echo N > /sys/kernel/debug/failslab/ignore-gfp-wait");
      sprintf(buf, "/proc/self/task/%ld/fail-nth", syscall(SYS_gettid));
      fail_nth = open(buf, O_RDWR);
      for (i = 1;; i++) {
              sprintf(buf, "%d", i);
              write(fail_nth, buf, strlen(buf));
              res = socketpair(AF_LOCAL, SOCK_STREAM, 0, fds);
              err = errno;
              pread(fail_nth, buf, sizeof(buf), 0);
              if (res == 0) {
                      close(fds[0]);
                      close(fds[1]);
              }
              printf("%d-th fault %c: res=%d/%d\n", i, atoi(buf) ? 'N' : 'Y',
                      res, err);
              if (atoi(buf))
                      break;
      }
```

```
        return 0;
}
```

An example output:

```
1-th fault Y: res=-1/23
2-th fault Y: res=-1/23
3-th fault Y: res=-1/12
4-th fault Y: res=-1/12
5-th fault Y: res=-1/23
6-th fault Y: res=-1/23
7-th fault Y: res=-1/23
8-th fault Y: res=-1/12
9-th fault Y: res=-1/12
10-th fault Y: res=-1/12
11-th fault Y: res=-1/12
12-th fault Y: res=-1/12
13-th fault Y: res=-1/12
14-th fault Y: res=-1/12
15-th fault Y: res=-1/12
16-th fault N: res=0/12
```