An efficient unit test and fuzz tools for kernel/libc porting

Bamvor Jian Zhang

Huawei

Oct, 6, 2016

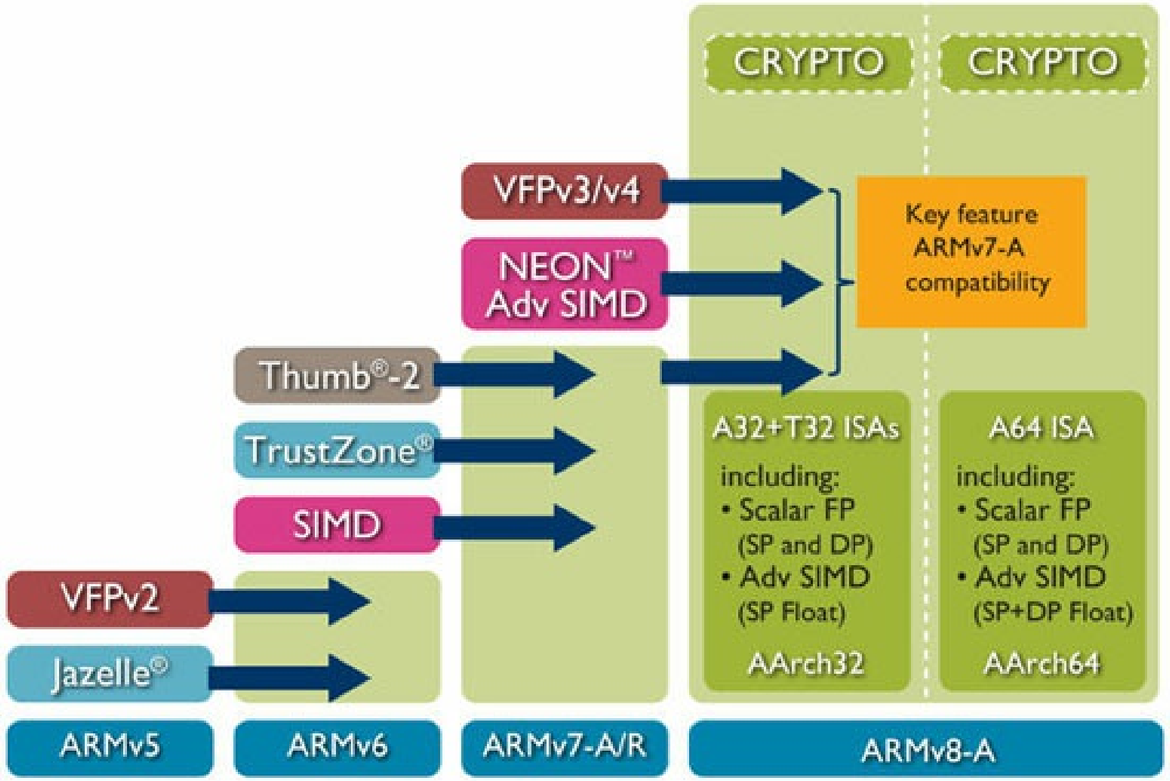
Self introduction

- Kernel developer from Huawei
- Linaro kernel working group assignee
- Focus on migration of 32-bit application
- Interested in memory management

aarch64 ILP32 overview

What is ILP32?

arm architecture

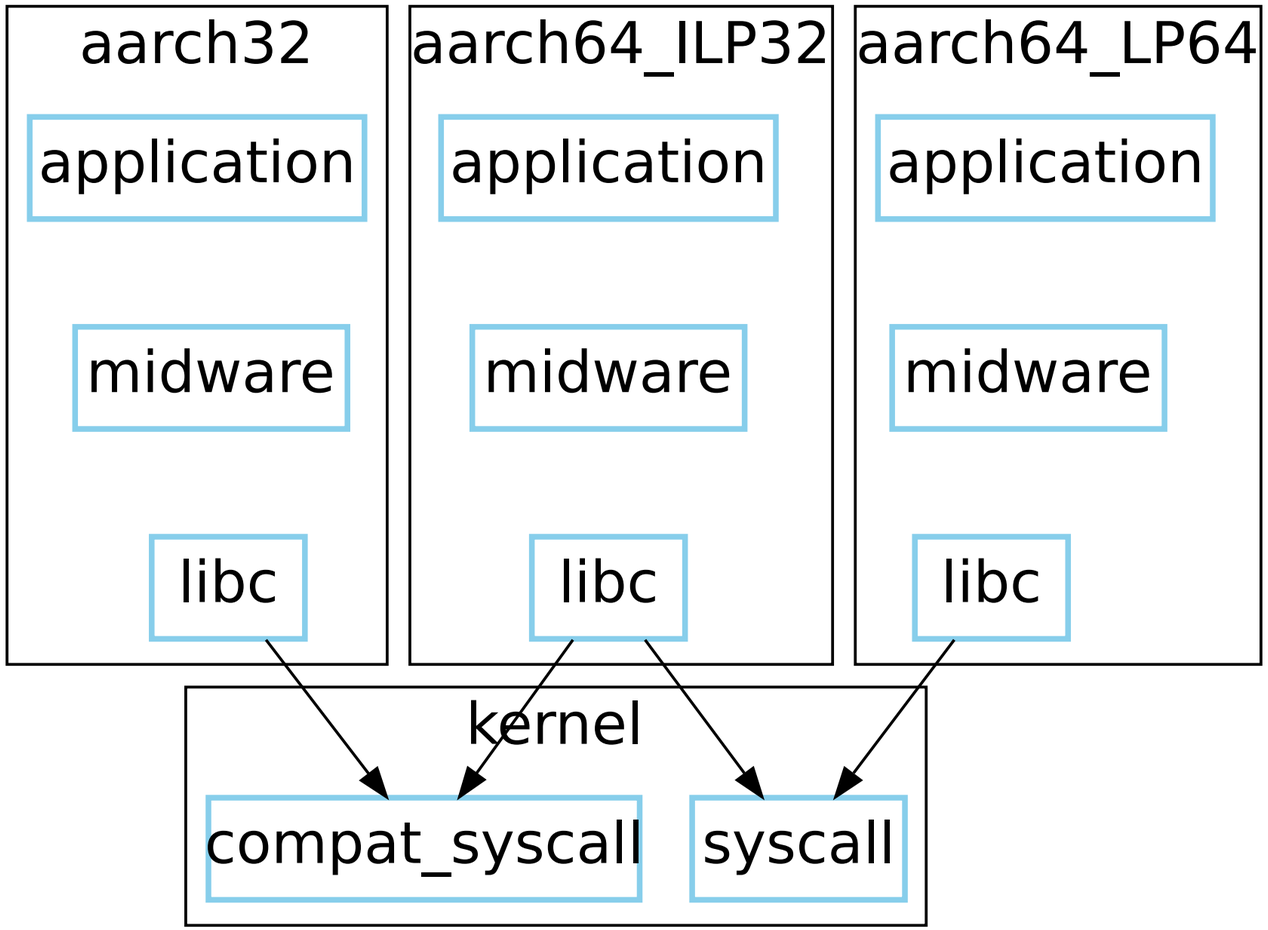


* This picture is belong to the ARM company

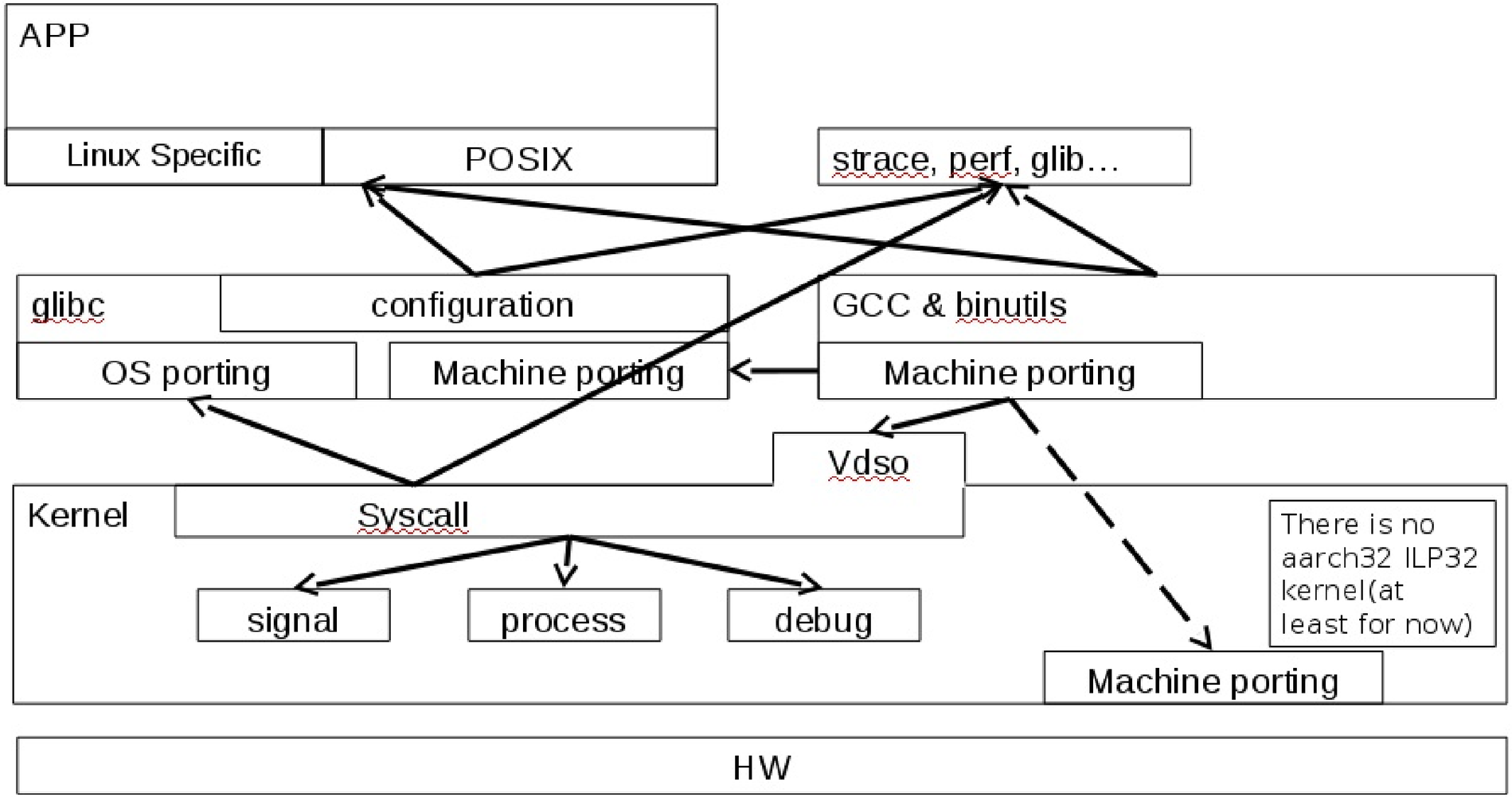
Data model

	ILP32	LP64	LLP64	ILP64
char	8	8	8	8
short	16	16	16	16
int	32	32	32	64
long	32	64	32	64
long long	64	64	64	64
size_t	32	64	64	64
pointer	32	64	64	64
	arm aarch32 aarch64 ILP32 x32 n32	aarch64 LP64	64bit windows	

Migrate 32-bit application to 64-bit hardware



ILP32 enablement



Why we need unit test for ILP32?

Lots of choices to be made for a new api

- The definition of basic type in userspace(NOT the kernel part!)
- Argument passing: one 64-bit register or two 32-bit registers
- Sanitize register contents

The definition of basic type in userspace

```
#define __DEV_T_TYPE          __UQUAD_TYPE
#define __UID_T_TYPE          __U32_TYPE
#define __GID_T_TYPE          __U32_TYPE
#define __INO_T_TYPE          __UQUAD_TYPE
#define __INO64_T_TYPE        __UQUAD_TYPE
#define __MODE_T_TYPE         __U32_TYPE
#define __NLINK_T_TYPE        __U32_TYPE
#define __OFF_T_TYPE          __SQUAD_TYPE
#define __OFF64_T_TYPE        __SQUAD_TYPE
#define __PID_T_TYPE          __S32_TYPE
#define __RLIM_T_TYPE         __UQUAD_TYPE
#define __RLIM64_T_TYPE       __UQUAD_TYPE
#define __BLKCNT_T_TYPE       __SQUAD_TYPE
#define __BLKCNT64_T_TYPE     __SQUAD_TYPE
#define __FSBLKCNT_T_TYPE     __UQUAD_TYPE
#define __FSBLKCNT64_T_TYPE   __UQUAD_TYPE
#define __FSFILCNT_T_TYPE     __UQUAD_TYPE
#define __FSFILCNT64_T_TYPE   __UQUAD_TYPE
```

The definition of basic type in userspace(Cont.)

```
#define __FSWORD_T_TYPE        __SWORD_TYPE
#define __ID_T_TYPE            __U32_TYPE
#define __CLOCK_T_TYPE         __SLONGWORD_TYPE
#define __TIME_T_TYPE          __SLONGWORD_TYPE
#define __USECONDS_T_TYPE      __U32_TYPE
#define __SUSECONDS_T_TYPE     __SLONGWORD_TYPE
#define __DADDR_T_TYPE         __S32_TYPE
#define __KEY_T_TYPE           __S32_TYPE
#define __CLOCKID_T_TYPE       __S32_TYPE
#define __TIMER_T_TYPE         void *
#define __BLKSIZE_T_TYPE       __S32_TYPE
#define __FSID_T_TYPE          struct { int __val[2]; }
/* ssize_t is always singed long in both ABIs. */
#define __SSIZE_T_TYPE         __SLONGWORD_TYPE
#define __SYSCALL_SLONG_TYPE   __SLONGWORD_TYPE
#define __SYSCALL_ULONG_TYPE   __ULONGWORD_TYPE
#define __CPU_MASK_TYPE        __ULONGWORD_TYPE
```

Four big changes in 3 years

Version A

- Most of syscalls are compat syscalls
- time_t and off_t are 32-bit

Version B

Similar to x32(x86 ILP32)

- Most of syscalls are 64-bit syscalls
- time_t and off_t are 64-bit
- Incompatible with arm32 compat-ioctl

Version C

Come back to version A

- Most of syscalls are compat syscalls
- time_t and off_t are 32-bit
- Pass 64-bit variable through one 64-bit reg
- Do the sign/zero extension when entering kernel

Version D

- More compat syscalls compare with aarch32
- Pass 64-bit variable through two 32-bit regs
- Clear the top-halves of of all the 64-bit regs of a syscall when entering kernel
- time_t is 32-bit and off_t is 64-bit

How many issues found by trinity when LTP syscall fails are < 20?

0

Compare existing kernel/glibc test tools

- Whether easy to reproduce a failure
- Whether support coverage
- Whether support libc test
- Whether generate full random data to basic data type

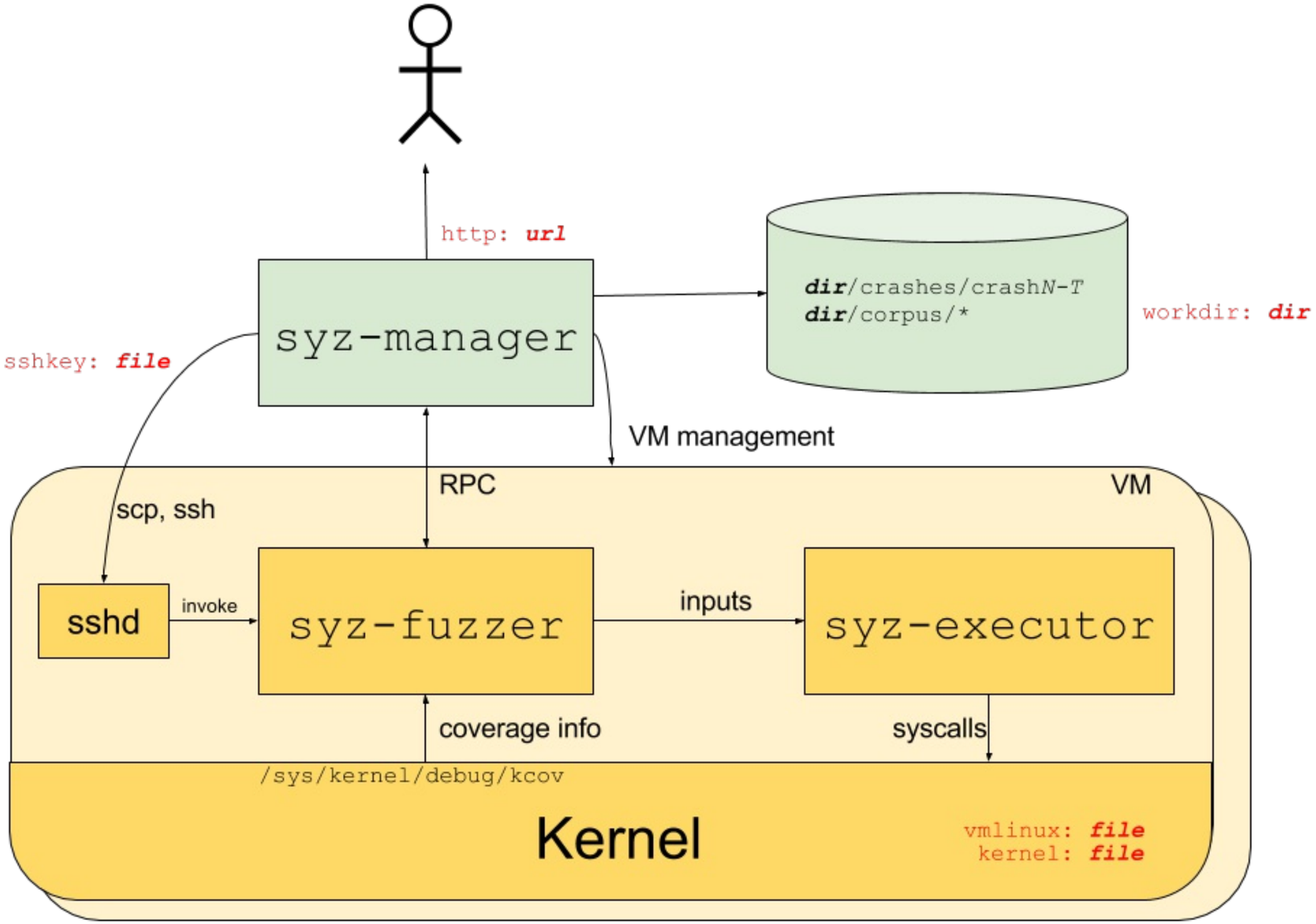
LTP and glibc testsuite

- The Classic testsuite for kernel and glibc
- Cons
  - No fuzz test. Test may pass while some issues are hidden

Trinity

- Pros
  - Generate fuzz data in a set of data type
  - Support lots of architecture
- Cons
  - Generate random address instead of basic data type for most of pointers
  - Takes too long to produce an issue and Takes much longer to re-produce and analyze it
  - Do not support coverage(?)

Syzkaller



* This picture is belong to the syzkaller project

Syzkaller(Cont.)

- Pros
  - Can recursively randomize base data type
  - Can generate readable short testcases
  - Can do the coverage
- Cons
  - Does not test C library

AFL and Triforce

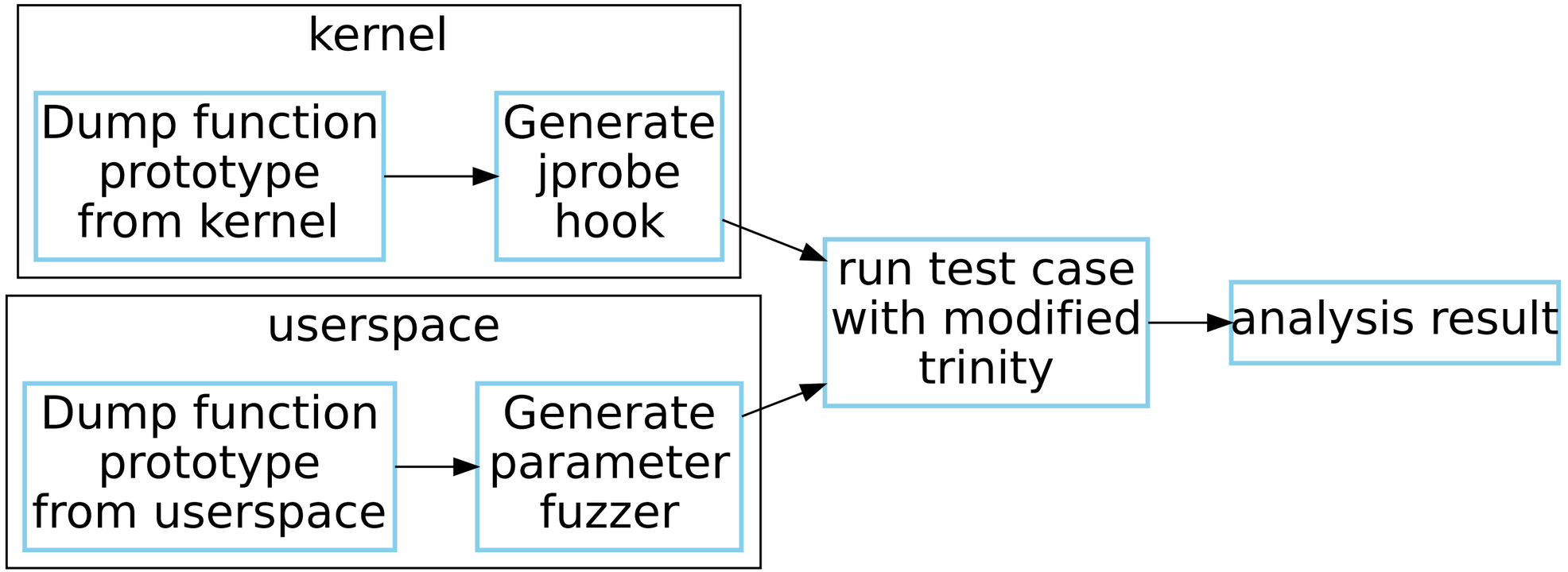
- Pros:
  - Base on the TriforceAFL
  - Do not need the coverage support in kernel
- Cons
  - Need special instruction in qemu

What's missing?

- No testsuite care about the porting of libc and kernel
- No full unit test for syscall

Introduce syscall unit test

The test flow of syscall unit test



Dump the prototype of function and struct

- Script base on abi-dumper
- Generate the fuzzer from json.

The fuzzer for structs in userspace

```
struct itimerspec *get_itimerspec()
{
    struct itimerspec *p = malloc(sizeof(struct itimerspec));

    p->it_interval.tv_sec = (unsigned long) rand64();
    p->it_interval.tv_nsec = (unsigned long) rand64();
    p->it_value.tv_sec = (unsigned long) rand64();
    p->it_value.tv_nsec = (unsigned long) rand64();

    //print all the value of this struct
    return p;
}
```

The Jprobe hook in kernel module

```
long JC_SyS_getitimer(int which, struct compat_itimerval *it)
{
    printk("parameter value:it<%u>, which<%u>", it, which);
    printk("it->it_interval.tv_sec<%u>, it->it_interval.tv_usec<%u>, it->it_value.tv_sec
            it->it_interval.tv_sec, it->it_interval.tv_usec,
            it->it_value.tv_sec, it->it_value.tv_usec);
    jprobe_return();        /* Always end with a call to jprobe_return(). */
    return 0;
}

static struct jprobe my_jprobe = {
    .entry = JC_SyS_getitimer,
    .kp = {
    .    symbol_name = "compat_sys_getitimer",
    },
};
```

```
static int __init jprobe_init(void)
{
    int ret;

    ret = register_jprobe(&my_jprobe);
    if (ret < 0) {
            printk(KERN_INFO "register_jprobe failed, returned %d\n", ret);
            return -1;
    }

    return 0;
}

static void __exit jprobe_exit(void)
{
    unregister_jprobe(&my_jprobe);
    printk(KERN_INFO "jprobe at %p unregistered\n", my_jprobe.kp.addr);
}
```

Modify trinity

- Call syscall through C library
- Add the missing struct in syscall
- Add jprobe hooks for capturing the arguments of syscall
- Add or Change some output message for script

Run it!

trinity/scripts/do_test_struct.sh

Found two issues in a specific version

- readahead
- sync_file_range

The return value test of syscall

- Random return value through kretprobe

TODO list

- Support all the syscalls which are not wrapped by libc
- Full automation in generating the fuzz code

What is the future of syscall unit test?

Contribute to LTP and/or glibc testsuite?

Or keep it as a standalone testsuite?

Code published in github

https://github.com/bjzhang/trinity/tree/syscall_unittest

https://github.com/bjzhang/abi-dumper/tree/json_output

Thanks