Zephyr system call argument marshaling war story

Nicolas Pitre
<npitre@baylibre.com>

2022-05-30

Nicolas Pitre

Zephyr OS System Call Mechanism Overview …

and how it may become a fight !

Zephyr Project

Zephyr Project

www.zephyrproject.org

- Small RTOS
- Monolithic kernel
- Privilege separation for threads

Zephyr Privilege Model

Zephyr Privileges Model

- Kernel Threads (privileged)
- User Threads (unprivileged)

Zephyr threads can be either.

Zephyr Privilege Model

Privilege Separation

- Memory Protection Unit (MPU)
- Processor Execution Level

Zephyr Privilege Model

Transition from user to privileged

- CPU interrupts
- Exception faults
- System calls

Zephyr System Calls

Privileged version of `k_sem_init()`

```
int k_sem_init(struct k_sem *sem,
               unsigned int initial_count,
               unsigned int limit)
{
        /*
         * Limit cannot be zero and count cannot be greater than limit
         */
        if (limit == 0 || limit > K_SEM_MAX_LIMIT ||
            initial_count > limit) {
                return -EINVAL;
        }

        sem->count = initial_count;
        sem->limit = limit;

        return 0;
}
```

Zephyr System Calls

Prototype declaration for `k_sem_init()`

```
__syscall int k_sem_init(struct k_sem *sem,
                         unsigned int initial_count,
                         unsigned int limit);
```

- The `__syscall` marker is the key.
- 400+ such syscalls exist at the moment.
- A Python script takes care of the rest.

Zephyr System Calls

Universal version of `k_sem_init()`

```
static inline int k_sem_init(struct k_sem * sem,
                             unsigned int initial_count,
                             unsigned int limit)
{
#ifdef CONFIG_USERSPACE
        if (z_syscall_trap()) {
                return arch_syscall_invoke3(
                               *(uintptr_t *) &sem,
                               *(uintptr_t *) &initial_count,
                               *(uintptr_t *) &limit,
                               K_SYSCALL_K_SEM_INIT);
        }
#endif
        compiler_barrier();
        return z_impl_k_sem_init(sem, initial_count, limit);
}
```

- The original `k_sem_init()` is renamed to `z_impl_k_sem_init()`.
- `z_syscall_trap()` checks if execution is unprivileged.
- `arch_syscall_invoke3()` does the privilege escalation.

Zephyr System Calls

RISC-V’s `arch_syscall_invoke3()`

```
static inline uintptr_t arch_syscall_invoke3(uintptr_t arg1, uintptr_t arg2,
                                             uintptr_t arg3,
                                             uintptr_t call_id)
{
        register ulong_t a0 __asm__ ("a0") = arg1;
        register ulong_t a1 __asm__ ("a1") = arg2;
        register ulong_t a2 __asm__ ("a2") = arg3;
        register ulong_t a7 __asm__ ("a7") = call_id;

        __asm__ volatile ("ecall"
                          : "+r" (a0)
                          : "r" (a1), "r" (a2), "r" (a7)
                          : "memory");
        return a0;
}
```

- The architecture-specific trap code uses register `a7` to branch to `z_mrsh_k_sem_init()`.

Zephyr System Calls

Argument marshalling for `k_sem_init()`

```
uintptr_t z_mrsh_k_sem_init(uintptr_t arg0, uintptr_t arg1, uintptr_t arg2,
                uintptr_t arg3, uintptr_t arg4, uintptr_t arg5)
{
        (void) arg3;    /* unused */
        (void) arg4;    /* unused */
        (void) arg5;    /* unused */
        int ret = z_vrfy_k_sem_init(
                          *(struct k_sem **)&arg0,
                          *(unsigned int*)&arg1,
                          *(unsigned int*)&arg2);
        return (uintptr_t) ret;
}
```

Zephyr System Calls

Syscall argument validation for `k_sem_init()`

```
int z_vrfy_k_sem_init(struct k_sem *sem,
                      unsigned int initial_count,
                      unsigned int limit)
{
        Z_OOPS(Z_SYSCALL_OBJ_INIT(sem, K_OBJ_SEM));
        return z_impl_k_sem_init(sem, initial_count, limit);
}
```

- Make sure `sem` points to a valid `struct k_sem` object.

Zephyr System Call Debugging

What about debugging with -O0

RISC-V (rv64) assembly output

```
static inline int k_sem_init(struct k_sem * sem, unsigned int initial_count, unsigned int limit)
{
    80000ad0:   6105                    addi    sp,sp,32
    80000ad2:   ec06                    sd      ra,24(sp)
    80000ad4:   e42a                    sd      a0,8(sp)
    80000ad6:   c22e                    sw      a1,4(sp)
    80000ad8:   c032                    sw      a2,0(sp)
        if (z_syscall_trap()) {
    80000ada:   b39ff0ef                jal     ra,80000612
    80000ade:   c911                    beqz    a0,80000af2
                return arch_syscall_invoke3(
                               *(uintptr_t *) &sem,
                               *(uintptr_t *) &initial_count,
                               *(uintptr_t *) &limit,
                               K_SYSCALL_K_SEM_INIT);
    80000ae0:   6522                    ld      a0,8(sp)
    80000ae2:   00413583                ld      a1,4(sp) ; !!
    80000ae6:   6602                    ld      a2,0(sp) ; !!
    80000ae8:   0b700693                li      a3,183
    [...]
```

Zephyr System Call Debugging

Let’s get rid of the pointer

```
 static inline int k_sem_init(struct k_sem * sem,
                              unsigned int initial_count,
                              unsigned int limit)
 {
 #ifdef CONFIG_USERSPACE
         if (z_syscall_trap()) {
                 return arch_syscall_invoke3(
-                               *(uintptr_t *) &sem,
-                               *(uintptr_t *) &initial_count,
-                               *(uintptr_t *) &limit,
+                               (uintptr_t) sem,
+                               (uintptr_t) initial_count,
+                               (uintptr_t) limit,
                                K_SYSCALL_K_SEM_INIT);
         }
 #endif
```

Zephyr System Call Debugging

RESULT

```
zephyr/include/generated/syscalls/kernel.h:
    In function 'k_sleep':
zephyr/include/generated/syscalls/kernel.h:95:3:
    error: aggregate value used where an integer was expected
 95 |  return arch_syscall_invoke1((uintptr_t)timeout, K_SYSCALL_K_SLEEP);
    |                              ^~~~~~~~~~~~~~~~~
```

```
typedef int64_t k_ticks_t;

typedef struct {
        k_ticks_t ticks;
} k_timeout_t;

int32_t k_sleep(k_timeout_t timeout);
```

Better Zephyr System Call Marshalling

`k_timeout_t` usage is all over the place!

Possible solutions:

- Filter `k_timeout_t` to reference `timeout.ticks`

But more stuff pops up:

- more single-member structure aggregates
- arrays
- and the infamous `va_list`

The Infamous `va_list`

- Can be a pointer
- Can be an array
- Can be a structure
- Can be a compiler internal abstraction

Only `va_copy()` is truly portable.

Zephyr System Call Universal Marshalling

The Ultimate Solution: unions

```
static inline int k_sem_init(struct k_sem *sem,
                             unsigned int initial_count,
                             unsigned int limit)
{
#ifdef CONFIG_USERSPACE
        if (z_syscall_trap()) {
                union { uintptr_t x; struct k_sem *val; } parm0 =
                    { .val = sem };
                union { uintptr_t x; unsigned int val; } parm1 =
                    { .val = initial_count };
                union { uintptr_t x; unsigned int val; } parm2 =
                    { .val = limit };
                return arch_syscall_invoke3(
                               parm0.x, parm1.x, parm2.x,
                               K_SYSCALL_K_SEM_INIT);
        }
#endif
```

Zephyr System Call Universal Marshalling

Argument marshalling for `k_sem_init()`

```c
uintptr_t z_mrsh_k_sem_init(uintptr_t arg0, uintptr_t arg1, uintptr_t arg2,
                uintptr_t arg3, uintptr_t arg4, uintptr_t arg5)
{
        (void) arg3;    /* unused */
        (void) arg4;    /* unused */
        (void) arg5;    /* unused */
        union { uintptr_t x; struct k_sem * val; } parm0;
        parm0.x = arg0;
        union { uintptr_t x; unsigned int val; } parm1;
        parm1.x = arg1;
        union { uintptr_t x; unsigned int val; } parm2;
        parm2.x = arg2;
        int ret = z_vrfy_k_sem_init(parm0.val, parm1.val, parm2.val);
        return (uintptr_t) ret;
}
```

The End

Questions?