Monolithic kernels vs. microkernels

More than 1 million lines of code

Monolithic kernel (e.g. Unix, Linux, Windows)

System Call

Should be able to crash

Microkernel (e.g. L4)

Inter-process communication

Really should not crash

c.f. The great Brian Bershad / Jochen Liedtke microkernel performance debate
seL4_Send(seL4_CPtr dest, seL4_MessageInfo_t msgInfo)

LIBSEL4_INLINE_FUNC void
seL4_Send(seL4_CPtr dest, seL4_MessageInfo_t msgInfo)
{
  arm_sys_send(seL4_SysSend, dest, msgInfo.words[0],
               seL4_GetMR(0), seL4_GetMR(1), seL4_GetMR(2), seL4_GetMR(3));
}

seL4
linux4/arch/include/arm/.sel4/arch/syscalls.h
static inline void
arm_sys_send(sel4_Word sys, sel4_Word dest, sel4_Word info_arg,
        sel4_Word mr0, sel4_Word mr1, sel4_Word mr2, sel4_Word mr3)
{
    register sel4_Word dest_ptr asm("r0") = dest;
    register sel4_Word info asm("r1") = info_arg;

    /* Load beginning of the message into registers. */
    register sel4_Word msg0 asm("r2") = mr0;
    register sel4_Word msg1 asm("r3") = mr1;
    register sel4_Word msg2 asm("r4") = mr2;
    register sel4_Word msg3 asm("r5") = mr3;

    /* Perform the system call. */
    register sel4_Word scno asm("r7") = sys;
    asm volatile(
        "swi 70"
        : "r" (dest_ptr), "r" (msg0), "r" (msg1), "r" (msg2),
        "r" (msg3), "r" (info)
        : "r"(scno)
    );
}

sel4
/home4/sel4/arch/include/arch32/sel4/arch32/syscall.h
<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervisor (SVC)</td>
<td>Entered on reset or SWI instruction</td>
</tr>
<tr>
<td>FIQ</td>
<td>Fast interrupt</td>
</tr>
<tr>
<td>IRQ</td>
<td>Normal interrupt</td>
</tr>
<tr>
<td>Abort</td>
<td>Memory access violations</td>
</tr>
<tr>
<td>Undef</td>
<td>Undefined instructions</td>
</tr>
<tr>
<td>System</td>
<td>Privileged mode, same registers as user mode</td>
</tr>
<tr>
<td>User</td>
<td>User mode (regular processes)</td>
</tr>
</tbody>
</table>
ARMv7-A exception vectors

- FIQ
- IRQ
- reserved
- Data abort
- Prefetch abort
- Software interrupt
- Undefined instruction
- Reset

- Hardware interrupts
- Address of the syscall handler
- Address to jump to at reset or boot
BEGIN_FUNC (arm-swि syscall)

/* Store CPSR and LR_svc on supervisor stack, which currently points at the end of the current thread's user context */
srsia #MODE_SUPERVISOR

/* Set the FaultInstruction address, which in ARM mode is the LR_svc - 4. * NOTE: This is completely wrong and broken in thumb mode. */
sub lr, lr, #4

/* Store FaultInstruction */
str lr, [sp, #(PT_FaultInstruction - PT_LR_svc)]

/* Stack all user registers */
stmd sp, {r0-r1}

/* Load the kernel's real stack pointer */
mrs p15, θ, sp, c13, c0, 4

/* Load system call number as a c_handle_syscall argument. r0 and r1 are passed unmodified (cptr and msgInfo) respectively. */
mov r2, r7
b c_handle_syscall

END_FUNC (arm-swि syscall)
void VISIBLE
c_handle_syscall(word_t cptr, word_t msgInfo, syscall_t syscall)
{
    NODE_LOCK_SYSCALL;
    c_entry_hook();

    if (unlikely(syscall < SYSCALL_MIN || syscall > SYSCALL_MAX)) {
        handleUnknownSyscall(syscall);
        restore_user_context();
        UNREACHABLE();
    } else {
        slowpath(syscall);
        UNREACHABLE();
    }
}

void NORETURN
slowpath(syscall_t syscall)
{
    handleSyscall(syscall);

    restore_user_context();
    UNREACHABLE();
}
exception_t
handleSyscall(syscall_t syscall)
{
    exception_t ret;
    irq_t irq;

    switch (syscall) {
    case SysSend:
        ret = handleInvocation(false, true);
        if (unlikely(ret != EXCEPTION_NONE)) {
            irq = getActiveIRQ();
            if (irq != irqInvalid) {
                handleInterrupt(irq);
                Arch_finaliseInterrupt();
            }
        }
        break;
    < ... >
    default:
        fail("Invalid syscall");
    }

    schedule();
    activateThread();

    return EXCEPTION_NONE;
}
void VISIBLE NORETURN restore_user_context(void)
{
    NODE_UNLOCK_IF_HELD;

    word_t cur_thread_reg = (word_t) NODE_STATE(ksCurThread);

    c_exit_hook();
    if (config_set(CONFIG_ARM_HYPervisor_SUPPORT)) {
        (_)
    } else {
        asm volatile("mov sp, %[cur_thread] \n"
           "lsmdb sp, {r8-lr} \n"
           "rfe\n"
        :
            "n\n"
        :
            "[cur_thread] "r" (cur_thread_reg + LR_svc + sizeof(word_t))
        :
            "";
    }

    UNREACHABLE();
}
/* This C function should be the first thing called from C after entry from * assembly. It provides a single place to do any entry work that is not * done in assembly for various reasons */
static inline void c_entry_hook(void)
{
    arch_c_entry_hook();
}

/* This C function should be the last thing called from C before exiting * the kernel (be it to assembly or returning to user space). It provides * a place to provide any additional instrumentation or functionality * in C before leaving the kernel */
static inline void c_exit_hook(void)
{
    arch_c_exit_hook();
}