Lab 13, interrupts, part 2

By ADMIN | *Published:* APRIL 14, 2013

Time to write a code to process interrupts.
All interrupts that happens in system use the vector of interrupts (8 of them) which is located at **0xffff0000** (high memory case). This vector contains addresses that CPU should pass execution to.

1.	+00	Reset
2.	+04	Undefined
3.	+08	SWI
4.	+0C	Prefetch abort
5.	+10	Data abort
6.	+14	Reserved
7.	+18	IRQ
8.	+1C	FIQ

In *main()* we have to call function called *trapinit()* to do all initializations of interrupts handling.

Example: when interrupt happens – say IRQ (number 7), then cpu will pass execution to **0xffff0000+18**. Our task is to initialize that place with codes that will pass execution to specific calls in kernel.

It is done in next way: in assembler file **intr.s** we create two functions:

```
TEXT vectors(SB), $-4
    MOVW    0x18(PC), PC    /* reset */
    MOVW    0x18(PC), PC    /* undefined */
    MOVW    0x18(PC), PC    /* SWI */
    MOVW    0x18(PC), PC    /* prefetch abort */
    MOVW    0x18(PC), PC    /* data abort */
    MOVW    0x18(PC), PC    /* reserved */
    MOVW    0x18(PC), PC    /* IRQ */
    MOVW    0x18(PC), PC    /* FIQ */

TEXT vtable(SB), $-4
    WORD    $_vsvc(SB)      /* reset, in svc mode already */
    WORD    $_vund(SB)      /* undefined, switch to svc mode */
    WORD    $_vsvc(SB)      /* swi, in svc mode already */
    WORD    $_vpab(SB)      /* prefetch abort, switch to svc mode */
    WORD    $_vdab(SB)      /* data abort, switch to svc mode */
    WORD    $_vsvc(SB)      /* reserved */
    WORD    $_virq(SB)      /* IRQ, switch to svc mode */
    WORD    $_vfiq(SB)      /* FIQ, switch to svc mode */

TEXT _vund(SB), $-4
    ...
```

Then in *trapinit()* we copy bytes of these functions to **0xffff0000**:

```
enum { Nvec = 8 }; /* # of vectors */
typedef struct Vpage0 {
    void    (*vectors[Nvec])(void);
    u32int  vtable[Nvec];
} Vpage0;

void trapinit(void) {
    Vpage0 *vpage0;
    /* set up the exception vectors */
    vpage0 = (Vpage0*)HVECTORS;
    memmove(vpage0->vectors, vectors, sizeof(vpage0->vectors));
    memmove(vpage0->vtable,  vtable,  sizeof(vpage0->vtable));
    ...
```

You may see that when execution passed to **0xffff0000+18** (HVECTORS+IRQ), then cpu does *MOVW 0×18(PC), PC*, which means to jump to address which is located in memory just **+0×18** above – where the vtable with addresses of our kernel routines *_virq()*

Then our 8 assembler routines to do initial logic of handling interrupts"

```
TEXT _vund(SB), $-4
    MOVM.DB [R0-R3], (SP)
    MOVW    $PsrMund, R0
    B       _vswitch

TEXT _vsvc(SB), $-4
    MOVW.W  R14, -4(SP)
    MOVW    CPSR, R14
    MOVW.W  R14, -4(SP)
    BIC     $PsrMask, R14
    ORR     $(PsrDirq|PsrDfiq|PsrMsvc), R14
    MOVW    R14, CPSR
    MOVW    $PsrMsvc, R14
    MOVW.W  R14, -4(SP)

    B       _vsaveu

TEXT _vpab(SB), $-4
    MOVM.DB [R0-R3], (R13)
    MOVW    $PsrMabt, R0
    B       _vswitch

TEXT _vdab(SB), $-4
    MOVM.DB [R0-R3], (R13)
    MOVW    $(PsrMabt+1), R0
    B       _vswitch

TEXT _vfiq(SB), $-4             /* FIQ */
    MOVM.DB [R0-R3], (R13)
    MOVW    $PsrMfiq, R0
    B       _vswitch

TEXT _virq(SB), $-4             /* IRQ */
    MOVM.DB [R0-R3], (R13)
    MOVW    $PsrMirq, R0

_vswitch: /* switch to svc mode */
```

You see that they are using 4 words of stack *[R0-R3]*. That I will show later in *trapinit()*, but idea that we can set different stack addresses for each interrupt types – that are those arrays we added to *Mach*: *ulong fiqstack[4]; ulong irqstack[4]; ulong abtstack[4]; ulong undstack[4];*

Later it switches to **svc** mode and cpu gets stack with address which is set for svc mode.

```
_vswitch:                       /* switch to svc mode */
    MOVW        SPSR,   R1      /* state of cpu, cpsr */
    MOVW        R14,    R2      /* return code */
    MOVW        SP, R3          /* stack */

    MOVW        CPSR,   R14
    BIC         $PsrMask, R14
    ORR         $(PsrDirq|PsrDfiq|PsrMsvc), R14
    MOVW        R14, CPSR       /* switch! */

    MOVW        R0, R0          /* gratuitous noop */

```

```
	MOVM.DB.W	[R0-R2], (SP)	/* set ureg->{type, psr, pc}; */
					/* SP points to ureg->type  */
	MOVW		R3, -12(SP)
	MOVM.IA		(R3), [R0-R3]	/* restore [R0-R3] from previous mode */

_vsaveu:
	MOVW.W		R11, -4(SP)	/* save link */

	SUB		$8, SP
	MOVM.DB.W	[R0-R12], (SP)

	MOVW		$setR12(SB), R12/* Make sure we've got the kernel's SB */
	MOVW		SP, R0		/* first arg is pointer to ureg */
	SUB		$(4*2), SP	/* space for argument+link (for debugger) */
	MOVW		$0xdeaddead, R11/* marker */
	BL		trap(SB)

_vrfe:					/* Restore Regs */
	MOVW		CPSR, R0	/* splhi on return */
	ORR		$(PsrDirq|PsrDfiq), R0, R1
	MOVW		R1, CPSR
	ADD		$(8+4*15), SP	/* [r0-R14]+argument+link */
	MOVW		(SP), R14	/* restore link */
	MOVW		8(SP), R0
	MOVW		R0, SPSR
	MOVM.DB.S	(SP), [R0-R14]	/* restore user registers */
	MOVW		R0, R0		/* gratuitous nop */
	ADD		$12, SP		/* skip saved link+type+SPSR*/
	RFE				/* MOVM.IA.S.W (SP), [PC] */
```

In codes above we can use *R0-R3* as we saved then in Mach object. With 4 registers we can perform disabling interrupts (svc), next CPU state saving into Ureg object, adjusting *SB* and finally pass the control to *trap()* C-routine of kernel to process the exception.

On return from *trap()* – **_vrfe**, it does opposite, restore registres from Ureg object, enables interrupts and pass control back to the point where execution was interrupted.

FILES:
intr.s
trap.c
mkfile

This entry was posted in *Blog*, *Inferno OS*, *Raspberry Pi*, *Research*. Bookmark the *permalink*. *Post a comment* or leave a trackback: *Trackback URL*.

« *Lab 12, interrupts, part 1*　　*Lab 14, interrupts, part 3* »