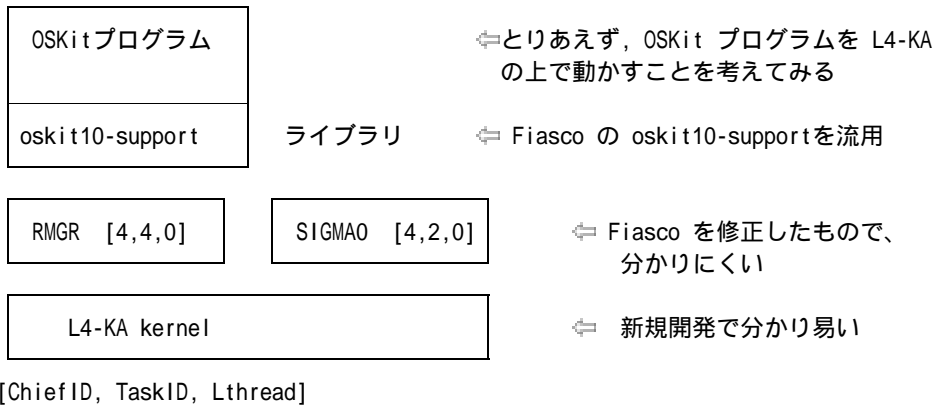


§ 位置付け



§ PC における割り込みベクタ番号と IRQ番号の対応

INT番号	IRQ番号	
00-01		例外ハンドラ
02		Non Maskable Interrupt
03-07		例外ハンドラ
08	IRQ-0	システムタイマ
09	IRQ-1	キーボード
0A	IRQ-2	IRQ-9 に接続される
0B	IRQ-3	シリアルポート
0C	IRQ-4	シリアルポート
0D	IRQ-5	ISA サウンドカード等
0E	IRQ-6	Floppy disc controller
0F	IRQ-7	パラレルポート
10-6F		ソフトウェア割り込み
70	IRQ-8	リアルタイムクロック
71	IRQ-9	IRQ-2 に接続される
72	IRQ-10	
73	IRQ-11	
74	IRQ-12	PS/2マウス
75	IRQ-13	数値演算 co processor
76	IRQ-14	Primary IDE controller
77	IRQ-15	Secondary IDE controller
78-FF		ソフトウェア割り込み

§ PCI: 8259チップのカスケード接続

(略)

§ L4-KAの IRQの処理設定

○ I4-ka/kernel/src/x86/interrupt.c

```
void init_irqs( )
```

- IRQの初期設定のヘンテコナ処理が書かれている。

§ L4-KAの IRQの処理の流れ

- 割り込みの発生

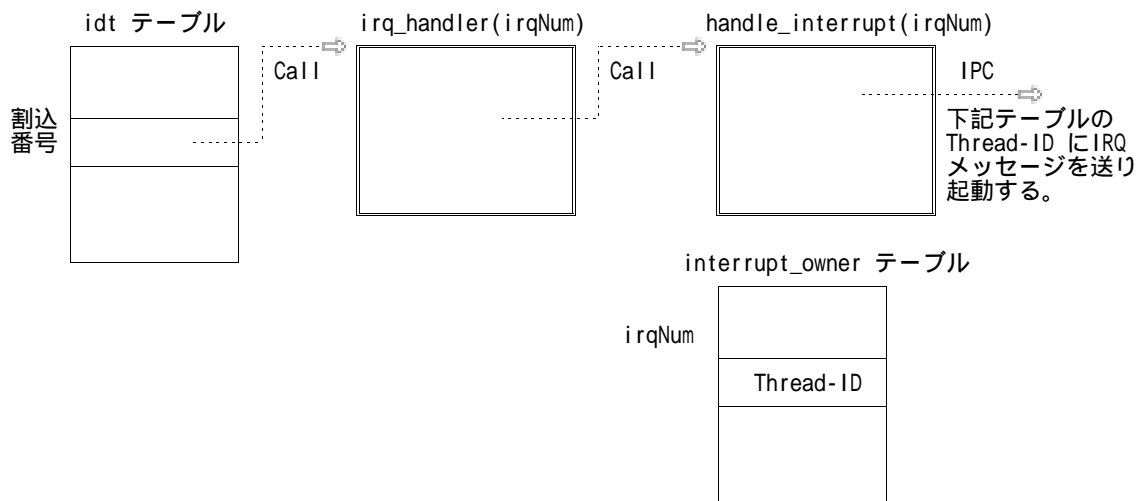
⇒ 割り込みベクタ番号で IDTテーブルをindex

```
CALL irq_handler (irq_num);
```

⇒ l4-ka/kernel/src/x86/interrupt.c :

```
void irq_handler (dword_t irq_num) を実行
```

⇒ l4-ka/kernel/src/x86/interrupt.c :



- interrupt_owner テーブルへの登録と削除

これも IPC で行う。

パラメータが『 rec_timeout=0, dest=1+irq』の IPCは、IRQ-association, つまり interrupt_owner テーブルへの登録 削除を行う。

IF interrupt_owner テーブルの該当entry がnullならば、IPC を実行したスレッドをここに登録する。

プログラム

```
l4-ka/kernel/src/ipc.c のなかの sys_ipc() 及び ipc_handle_kernel_id( )
```

§ RMGR での IRQ処理

RMGR のタスク番号

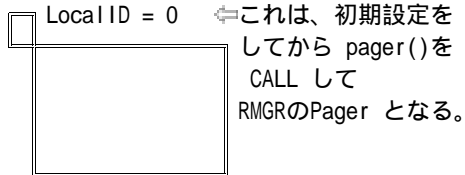
```
TaskID = 4
```

```
ChiedID = 4
```

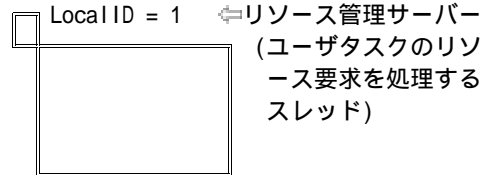
関連スレッドとそれが実行する関数

RMGR の中では、以下のスレッドが走る。

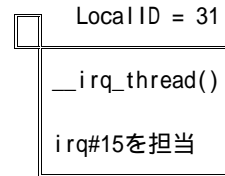
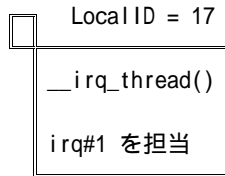
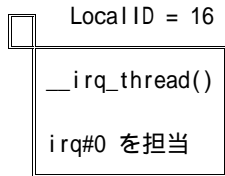
RMGR のメインスレッド



リソース管理スレッド



IRQスレッド ⇐ L4-KA では、使う必要がないか？



○ IRQスレッドは、関数 __irq_thread(int irqNum) を実行している。

○ PROC __irq_thread(int irqNum);

CALL irq_attach(irqNum); // 自IRQ スレッドを登録する。

LOOP

RECV IRQ-associate メッセージを受信する (要求スレッド=)

要求スレッド を本IRQ にassociate する。

SEND 完了メッセージ。

LOOPEND;

END;

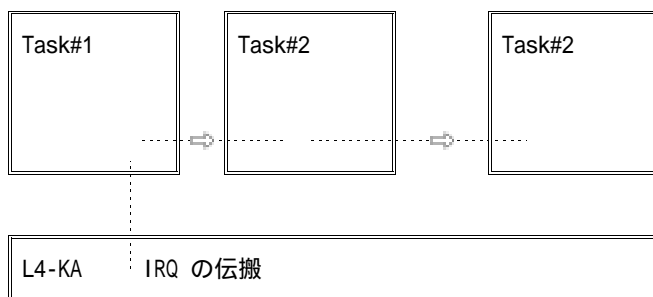
ユーザスレッドを IRQにassociate する機構

○rmgr/rmgr.c の関数 rmgr() の中で、上記 IRQスレッドに IRQ-associateメッセージを送信。

ipc_call (irq# のIRQスレッド、0, 1, 0, ...)

§ 検討課題

(1) IRQ処理を行う複数タスクの扱い



irq sharing

(2) I/O空間のアクセス権

ユーザモードで走るタスクから I/O 空間を操作するには? ⇒ TSSのI/O-bitmap

(3) 各種サービスシステム (ex. ファイルシステムやネットワークシステム) は、

(a) iOSKit を利用するか、それとも (b) Linux から切り出してもろにつかうか ?

⇒ (a)の場合は、Fiascoのoskit10_supportライブラリ相当を作る。

(4) L4-KA HazelNutsの稼働状況

○VMware上では動作するが、実機上では動作しない

○make xconfig にて"x86-Architecture specific fpage functions"=yes とすると動作しない。

プログラムロジックはこちらの方が簡明。

○make xconfig にて"Enabling new mapping DB"=yes とすると動作しない。

○grubのmenu.lstに"modaddrt=0x03000000" を指定すると動作しない。 32M ?

Fiasco の oakit10_support ライブラリ

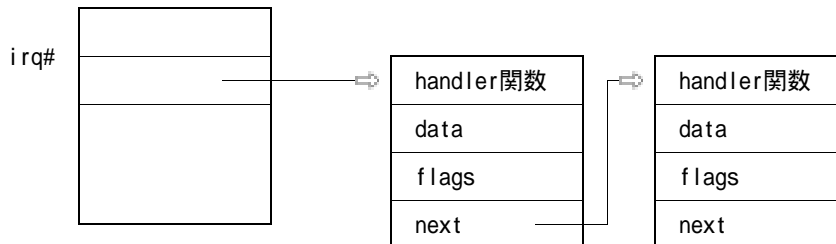
§ oskit10_support(): From Fiasco

===== fiasco/l4/oskit10_support/lib/l4ka-src/irq.c =====

○これは Fiasco 上で OSKitを動かすためのライブラリー関数であり、ユーザプログラムにリンクして使う。

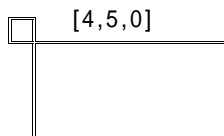
irq_handlerの登録: IRQ ハンドラを登録する仕組みを、下図に示す。

```
static struct irqaction *irq_handler[16];
```

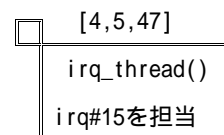
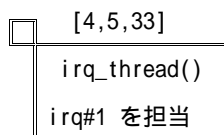
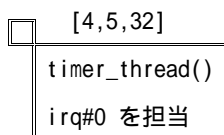


本ライブラリーの中で使われるスレッド

User taskのメインスレッド



IRQスレッド



[注] irq.c: static unsigned irq_thread_no_offset = 0x20; に修正する。(元は 0x40)

上記 IRQスレッドは、次のプログラムを実行している。

```
PROC irq_thread(int irq)
    RMGR経由で IRQ associateする。
    IPC_RECV (1+irq, 0, &dummy, &dummy, rec_timeout=0, &dummysdope);
    // 本スレッドを IRQ associateする。
    LOOP
        IPC_RECV (1+irq, 0, &dummy, &dummy, L4_IPC_NEVER, &dummysdope);
        //IRQ メッセージを待つ
        CALL oskit_support_irq_wakeup(irq, 0);
    ENDLLOOP;
END irq_thread;
```

○ FUNC oskit_support_irq_wakeup(int irq, unsigned ret_eip): int;
 //これがまたへんてこなロジックのプログラムであるが、要点は
 //結局は call_handlers() を起動すること。

```

.....
CALL call_handlers(irq);
.....
END;

```

```

○ FUNC call_handlers(int irq);
  FOR a = irq_handler[irq]; a; a = a->next DO
    CALL a->handler (a->data);      ⇐ IRQ に対応したハンドラを実行
  OD;
END;

```

IRQハンドラを登録する関数

```
int osenv_irq_alloc (int irq, void (*handler)(void*), void *data, int flags);
```

【論理】

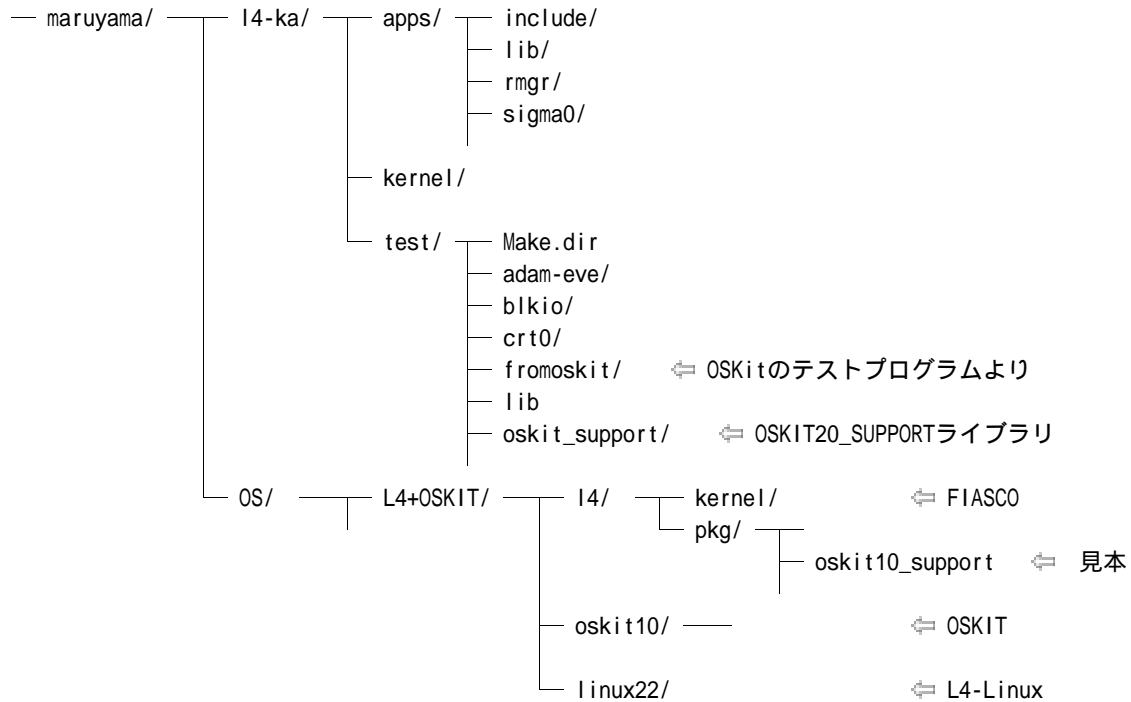
```

FUNC osenv_irq_alloc( ....);
  osenv_intr_disable();
  IF irq_handler[irq]が登録済で、且つ OSENV_IRQ_SHARABLE で無い
    THEN intrを基に戻し、エラーを返す。
  FI;
  irq_handler[irq]にハンドラを登録する。
  IF このirq を担当する IRQ_THREAD が未生成
    THEN IF irq==0 THEN IRQ_THREAD#0 を生成し、timer_thrad() を実行させる。
          ELSE IRQ_THREAD#irq を生成し、irq_thrad() を実行させる。
    FI;
  FI;
  osenv_irq_enable(irq);
  osenv_intr_enable();
  RETURN rcode;
ENDFUNC;

```

L4-KA + oskit20_supportの上で oskitのテストプログラムを動かす

【Directory】



§ oskitライブラリよりもoskit20_support ライブラリを優先してリンクするには?

- l4-ka/test/fromoskit/Makefile を参考
- 現在は zzz.cファイルの中から強制的にoskit20_support ライブラリの関数を呼び出しているが、リンカーのマニュアルを調べれば正当な手法があるであろう。

§ Fiasco 用のソースプログラムを L4-KAにて(修正量最少にて) 使う方法

- L4 のprimitivesを呼ぶ関数は、ヘッダーファイルにてインライン定義されている。
- Fiasco用のソースプログラムで Fiasco のヘッダーファイルをインクルードしている所を、L4-KA の l4-ka/apps/include からインクルードするように修正する。
勿論、場合によっては更なる修正も必要。

§ メモリ管理

- Fiasco + OSKit では、osKit10_support ライブラリーでメモリの初期設定をしている。
- L4-KA の場合は、 fromoskit/pingreply2.c を参照
⇒ プログラムの先頭で、以下の setup_lmm() を呼んでおく。

【Program】

```

#include <oskit/lmm.h>
extern lmm_t malloc_lmm;
#define MPOOLSIZE (8*1024*1024)
static char mempool[MPOOLSIZE];
static lmm_t lmm;
  
```

```
static lmm_region_t region;

void setup_lmm()
{ char * mem;
  mem = (char *)mempool;
  lmm_init(&lmm);
  lmm_add_region(&lmm, &region, 0, -1, 0, 0);
  lmm_add_free(&lmm, mem, MPOOLSIZE);
  malloc_lmm = lmm;
}
```

§ カーネルブレイク

- KBREAK("....."); MBREAK(".....");
- l4-ka/test/fromoskit/kdebug.h を参照

§ プログラムの修正

- foo.c ⇒ foo.c.ORG としてオリジナルファイルは保存しておく。
- 修正箇所

```
//K.M. //KM 01/01/23
//UM 01/01/23
```