Model przerwań FreeBSD 5.x

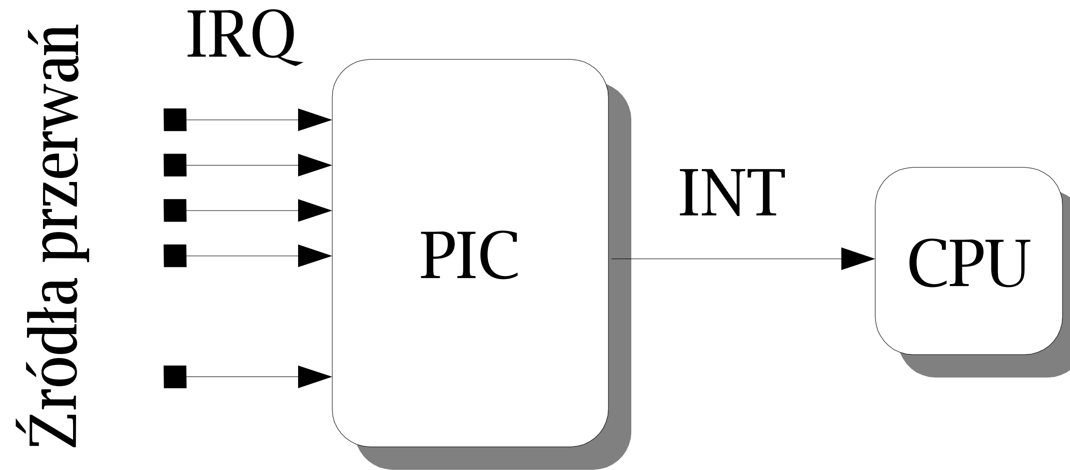
Rafał Jaworowski
raj@semihalf.com

MeetBSD 2005, Kraków 17-19 czerwca 2005

Wstęp

- Zakres i profil wykładu
  - kontekst
  - perspektywa programisty
  - infrastruktura
    - zależna od architektury, sprzętu
    - niezależna
- Technika obsługi niezależnych zdarzeń
  - przerwania
  - polling (próbkowanie)

Kontekst



- Fizyczne źródła przerwań
- Kontroler przerwań (Programmable Interrupt Controller)
- CPU

Definicje

- przerwanie
  - sygnał elektryczny
  - wiadomość
- zbliżone terminy
  - wyjątek
  - pułapka

Kontekst

- nacisk na omówienie infrastruktury zewnętrznych przerwań sprzętowych
  - brak szczegółowych porad np. jak napisać handler
- ogólny problem: droga od pojawienia się fizycznego sygnału przerwania do wywołania określonej funkcji jego obsługi
- dwa rodzaje przerwań: H/W i S/W
  - S/W nie są związane z fizycznym wejściem na kontrolerze PIC, ale obsługiwane jak H/W
- przykłady dla ATPIC (kaskada 8259), AMD64, FreeBSD 5.4-RELEASE

Programowanie sterownika

- sterownik żąda przydzielenia przerwania dla urządzenia
  - skąd znamy IRQ?
- w czasie mi_startup() - SI_SUB_CONFIGURE
  - podczas autokonfiguracji (probe/attach) następuje przydzielenie zasobów dla urządzenia m.in. przerwania

Konfiguracja przerwania

- np. pci/if_de.c (tulip)

```
...
rid = 0;

res = bus_alloc_resource_any(dev, SYS_RES_IRQ, &rid,
                             RF_SHAREABLE | RF_ACTIVE);

if (res == 0 || bus_setup_intr(dev, res, INTR_TYPE_NET,
                               intr_rtn, sc, &ih)) {

      ...

}
```

- BUS_SETUP_INTR = nexus_setup_intr()

```
nexus_setup_intr() {

    ...
    error = intr_add_handler(device_get_nameunit(child),
          rman_get_start(irq), ihand, arg, flags, cookiep);
    ...

}
```

intr_add_handler()

- na podstawie nr-u wektora odnajduje strukturę opisującą źródło przerwania
- dodaje handler
  - ithread_add_handler()
  - ustawia priorytet, wstawia na koniec kolejki it_handlers
- w efekcie dla danego fizycznego źródła przerwania mamy podłączoną funkcję jego obsługi
- potencjalnie jedno źródło przerwań może mieć wiele handler-ów

Inicjalizacja tablicy przerwań

- tablica deskryptorów przerwań/wyjątków
  - Interrupt Descriptor Table
  - dla wszystkich wektorów przerwań H/W na początek handler domyślny *rsvd*
  - indywidualne handler-y dla wyjątków procesora
- ustawienie rejestru bazowego przerwań
  - dla x86 – IDTR (Interrupt Descriptor Table Register) wskazuje na tablicę IDT

Inicjalizacja c.d.

- obsługa przerwań (w CPU) jeszcze wyłączona!
- wołane b. wcześnie, przed mi_startup(), podczas inicjalizacji specyficznej dla maszyny np. w hammer_time() lub init386()
- w efekcie ustawione są wektory dla wyjątków CPU, wektory przerwań H/W i S/W mają domyślny handler *rsvd*

Wczesna inicjalizacja kontrolera PIC

- atpic_startup()
  - wołane "ręcznie", poza autokonfiguracją NEWBUS-a (zanim hierarchia NB w ogóle istnieje)
  - inicjalizuje kontroler(y) PIC
- tablice kontrolerów i źródeł przerwań
  - atpics[]
  - atintrs[], element odpowiada jednemu wejściu PIC
- parametry wyzwolenia przerwań itp.

Inicjalizacja kontrolera PIC c.d.

- w efekcie zainicjowane kontrolery PIC
  - większość zródeł jest jeszcze zablokowana w PIC (!= CPU)
- uzupełniona inicjalizacja tablicy IDT dla indeksów przerwań H/W
  - niskopoziomowe handler-y Xatpic_intr0 .. Xatpic_intr15

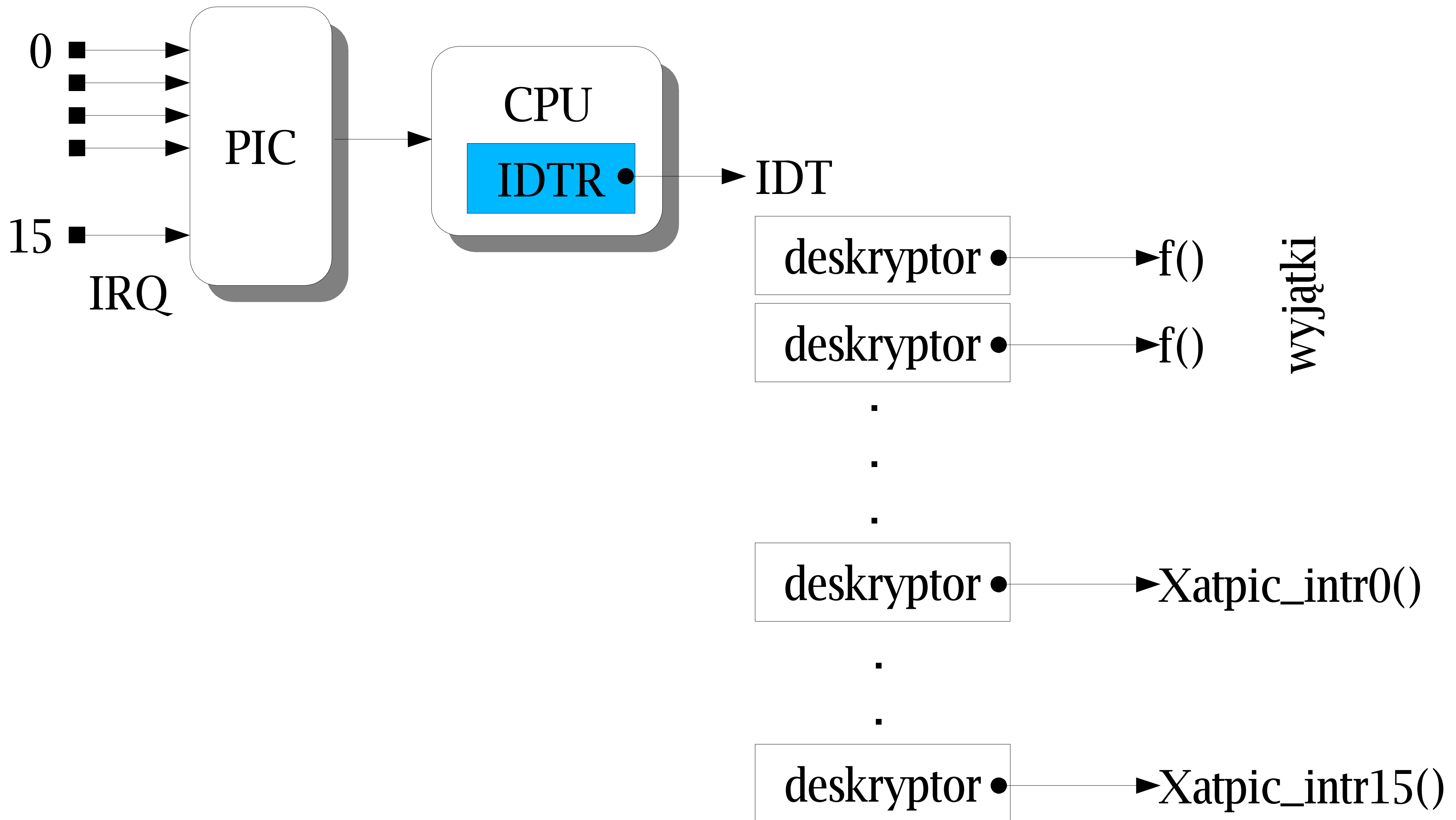
Niskopoziomowy handler Xatpic_intr0..15

- amd64/isa/atpic_vector.S

```
        ...
IDTVEC(vec_name) ;
        subq    $TF_RIP,%rsp ;
        testb   $SEL_RPL_MASK,TF_CS(%rsp) ;
        jz      1f ;
        swapgs ;
1:      movq    %rdi,TF_RDI(%rsp) ;
        movq    %rsi,TF_RSI(%rsp) ;
        ...

        movq    %r10,TF_R10(%rsp) ;
        movq    %r11,TF_R11(%rsp) ;
        movq    %r12,TF_R12(%rsp) ;
        movq    %r13,TF_R13(%rsp) ;
        movq    %r14,TF_R14(%rsp) ;
        movq    %r15,TF_R15(%rsp) ;
        FAKE_MCOUNT(TF_RIP(%rsp)) ;
        movq    $irq_num, %rdi;         /* pass the IRQ */
        call    atpic_handle_intr ;
        MEXITCOUNT ;
        jmp     doreti
```

Obrrr..az zależności



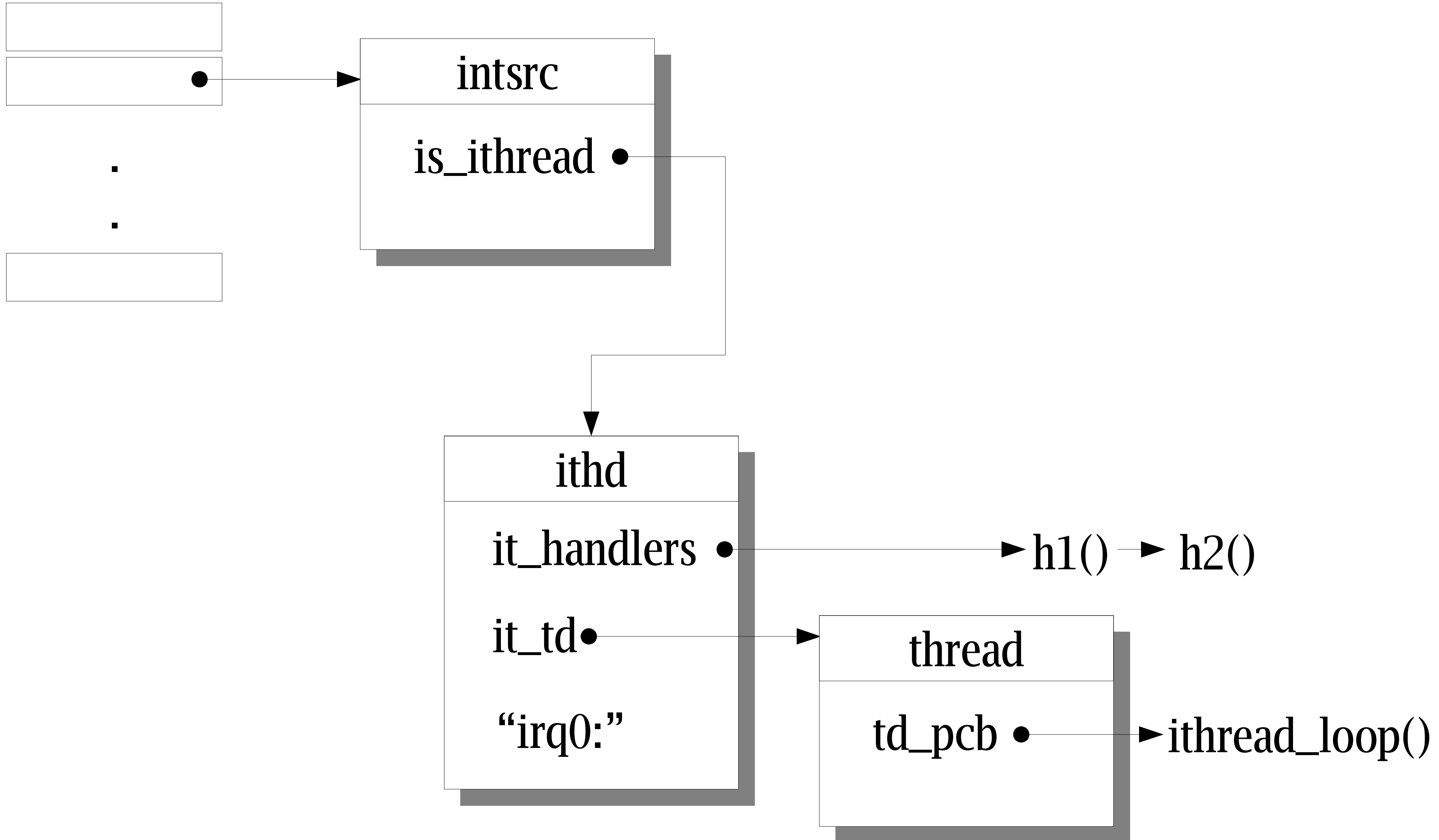
Inicjalizacja niezależna od sprzętu

- moduły SYSINIT/SI_SUB_INTR
  - podczas mi_startup()
  - przed SI_SUB_DRIVERS i SI_SUB_CONFIGURE (autokonfiguracja urządzeń)
- intr_init()
  - przypisuje nazwy przerwań – intrnames[]
  - ustanowienie mutexa dla tablicy przerwań
- atpic_init()
  - w pętli dla wszystkich źródeł przerwań (16) woła intr_register_source()

intr_register_source()

- rejestruje źródło przerwań w globalnej tablicy interrupt_sources[] - wpisuje do niej adres struktury opisującej źródło
  - tworzy strukturę opisującą wątek przerwania dla źródła
    - ithread_create()
    - przypisuje nazwę "irq%d:"
    - tworzy sam wątek
      - główna funkcja – ithread_loop()
  - tworzy i przypisuje nazwę przerwania (na podst. nazwy wątku związanego ze źródłem
  - inicjalizuje obsługę zbłąkanego przerwania (stray)

intr_register_source() c.d.



ithread_loop()

- dostaje adres do struktury opisującej wątek przerwania, z którą jest związany
- w pętli, jeśli flaga ithd->it_need
  - dla wszystkich handler-ów z listy it_handlers
    - woła handler ih->ih_handler(ih->ih_argument)
- wykrywa ew. interrupt storm => wtedy zasypiamy do następnego tick-u hardclock-a
- dobrowolnie oddaje CPU

Wątki przerwań

```
PID  TT  STAT      TIME COMMAND
 ..
 ..
 12  ??  WL     0:00,00 [irq0: clk]
 13  ??  WL     0:55,79 [irq1: atkbd0]
 14  ??  WL     0:00,00 [irq3: sio1]
 15  ??  WL     0:00,00 [irq4: sio0]
 16  ??  WL     0:00,00 [irq5:]
 17  ??  WL     0:00,00 [irq6: fdc0]
 18  ??  WL     0:00,00 [irq7: ppc0]
 19  ??  WL     0:00,00 [irq8: rtc]
 20  ??  WL     0:03,53 [irq9: acpi0]
 21  ??  WL     0:00,00 [irq10:]
 22  ??  WL     0:18,17 [irq11: cbb0 cbb1++]
 23  ??  WL     0:47,51 [irq12: psm0]
 24  ??  WL     0:00,00 [irq13:]
 25  ??  WL     0:17,17 [irq14: ata0]
 26  ??  WL     0:00,07 [irq15: ata1]
 27  ??  WL     0:17,65 [swi1: net]
 28  ??  WL    25:35,27 [swi5: clock sio]
 29  ??  WL     0:00,00 [swi4: vm]
 ..
 ..
```

50 μs z życia przerwania

- wyzwolenie przerwania – linia wejściowa kontrolera PIC
- przerwanie do CPU, potwierdzenie, przekazanie nr-u wektora
- sterowanie przekazywane do niskopoziomowego handler-a przerwania Xatpic_intrN()
- handler ustanawia ramkę przerwania i woła wysokopoziomową funkcję obsługi atpic_handle_intr(irq, iframe) z numerem zgłaszanego przerwania i ramką

atpic_handle_intr()

- ustalenie adresu struktury opisującej źródło przerwania
- intr_execute_handlers()
  - statystyka liczby wystąpienia przerwania
  - jeżeli pierwszy handler z listy jest IH_FAST (czyli przerwanie typu INTR_FAST)
    - blokada przerwań w CPU - critical_enter()
    - w pętli dla wszystkich handler-ów z listy it_handlers, wywołanie po kolei każdego
    - sekwencja EOI (End Of Interrupt)
    - odblokowanie przerwań w CPU - critical_exit()

atpic_handle_intr() c.d.

- jeżeli przerwanie nie jest INTR_FAST
  - sekwencja EOI
  - zaszergowanie wątku związanego z przerwaniem – ithread_schedule(ithd); w konsekwencji wątek ten wstawiany jest do kolejki runqueue i zostanie zaszeregowany do wykonania
- dla przerwań INTR_FAST (z handler-ami IH_FAST) struktury wątku są stworzone, ale sam wątek nie jest nigdy szeregowany, zawsze handler-y z listy są wykonywane w kontekście przerwania

Przykłady przerwań

- INTR_FAST
  - zegarowe, npx, fdc, sio
- nie-FAST
  - prawie wszystkie inne, np. kart sieciowych

Podsumowanie

- dwa rodzaje przerwań H/W: FAST i pozostałe
- b. ciężki model
  - Xatpic_intrN() → atpic_handle_intr() → intr_execute_handlers(), potem:
    - FAST – zawołanie po kolei wszystkich handler-ów z listy
    - nie-FAST – zaszeregowanie wątku przerwania, który zostanie wykonany w swoim kontekście kiedyś później (kiedy?)
- nie do zastosowań czasu rzeczywistego
  - brak przewidywalności zwłoki
- referencje
  - man 9 kthread, ithread, swi_add, taskqueue

Model przerwań FreeBSD 5.x

Rafał Jaworowski
raj@semihalf.com

Dziękuję za uwagę

http://www.semihalf.com/pub/meetbsd/2005_przerwania.pdf

MeetBSD 2005, Kraków 17-19 czerwca 2005