

Ch 3. Process: The Principal Model of Execution (cont'd)

Nov. 7, 2006

Hyun-koo Jee

Senior researcher, DWE

Contents

3.0. Intro.

3.1. Introducing Our Program

3.2. Process Descriptor

3.3. Process Creation: `fork()`, `vfork()`, & `clone()`

3.4. Process Lifespan

3.5. Process Termination

3.6. Keeping Track of Process: Basic Scheduler
Construction

3.7. Wait Queues

3.8. Asynchronous Execution Flow

Contents

3.0. Intro.

3.1. Introducing Our Program

3.2. Process Descriptor

3.3. Process Creation: `fork()`, `vfork()`, & `clone()`

3.4. Process Lifespan

3.5. Process Termination

**3.6. Keeping Track of Process: Basic
Scheduler Construction**

3.7. Wait Queues

3.8. Asynchronous Execution Flow

3.6. Keeping Track of Processes

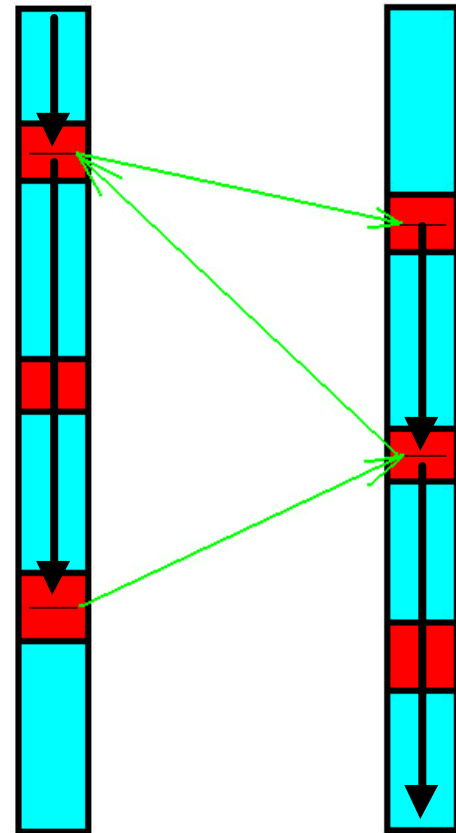
- Scheduling ?

— “

”

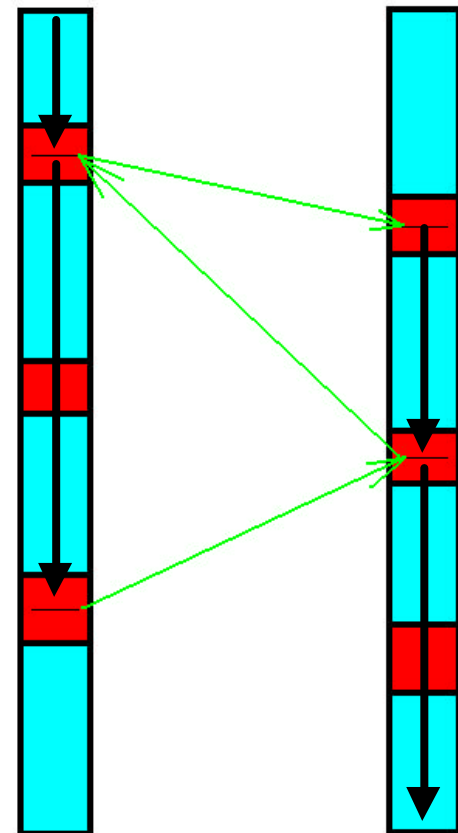
...

- Kernel
schedule()



3.6. Keeping Track of Processes

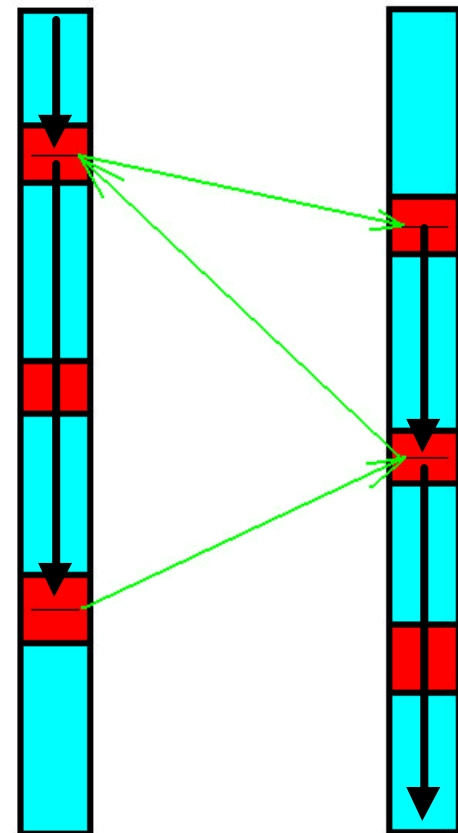
- `schedule()` 1
 - `set_tsk_need_resched()`
 - `current->thread_info->flags`
`TIF_NEED_RESCHED`
 가 set
 (가
 - `schedule()`
 (arch/*/kernel/entry.S)



3.6. Keep

- 가 time slice scheduler_tick()
- 가 , try_to_wake_up()

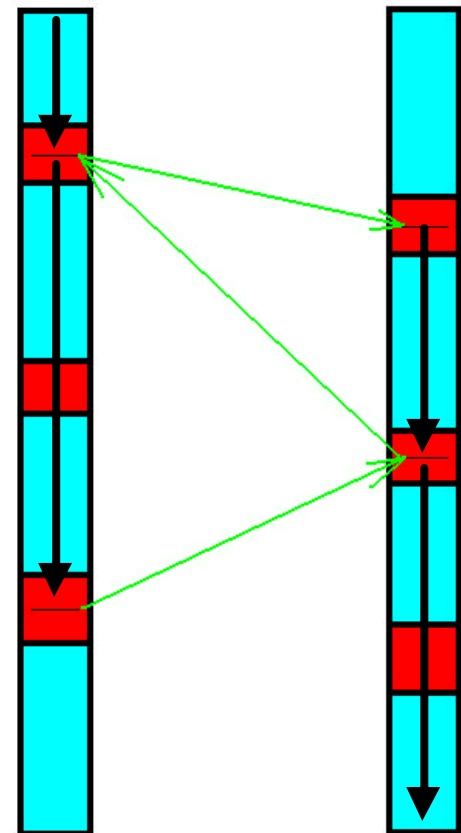
- schedule()
 - set tsk need resched()
 - current->thread_info->flags TIF_NEED_RESCHED
가 set .
(가)
 - , 가
 - , schedule()
(arch/*/kernel/entry.S)



3.6. Keeping Track of Processes

- `schedule()` 2

— , 가
,
`schedule()`
.



3.6. Keeping Track of Processes

- runqueue ?
 - “ , ”
“CPU가 .”
...
– 1 per CPU
– the most important data structure for scheduling

struct runqueue

prio_array_t *active

spinlock_t lock

task_t *curr, *idle

prio_array_t *expired

unsigned long nr_running

init best_expired_prio

prio_array_t arrays[0]

int nr_active

unsigned long bitmap [BITMAP_SIZE]

struct list_head queue [MAX_PRIO]

0

1

2

3

4

5

6

138

139

140

X

Y

...

Z

(queue of priority 138 task pointers)

A

B

...

C

(queue of priority 0 task pointers)

prio_array_t arrays[1]

int nr_active

unsigned long bitmap [BITMAP_SIZE]

struct list_head queue [MAX_PRIO]

0

1

2

3

4

5

6

138

139

140

U

V

...

W

(queue of priority 138 task pointers)

D

E

...

F

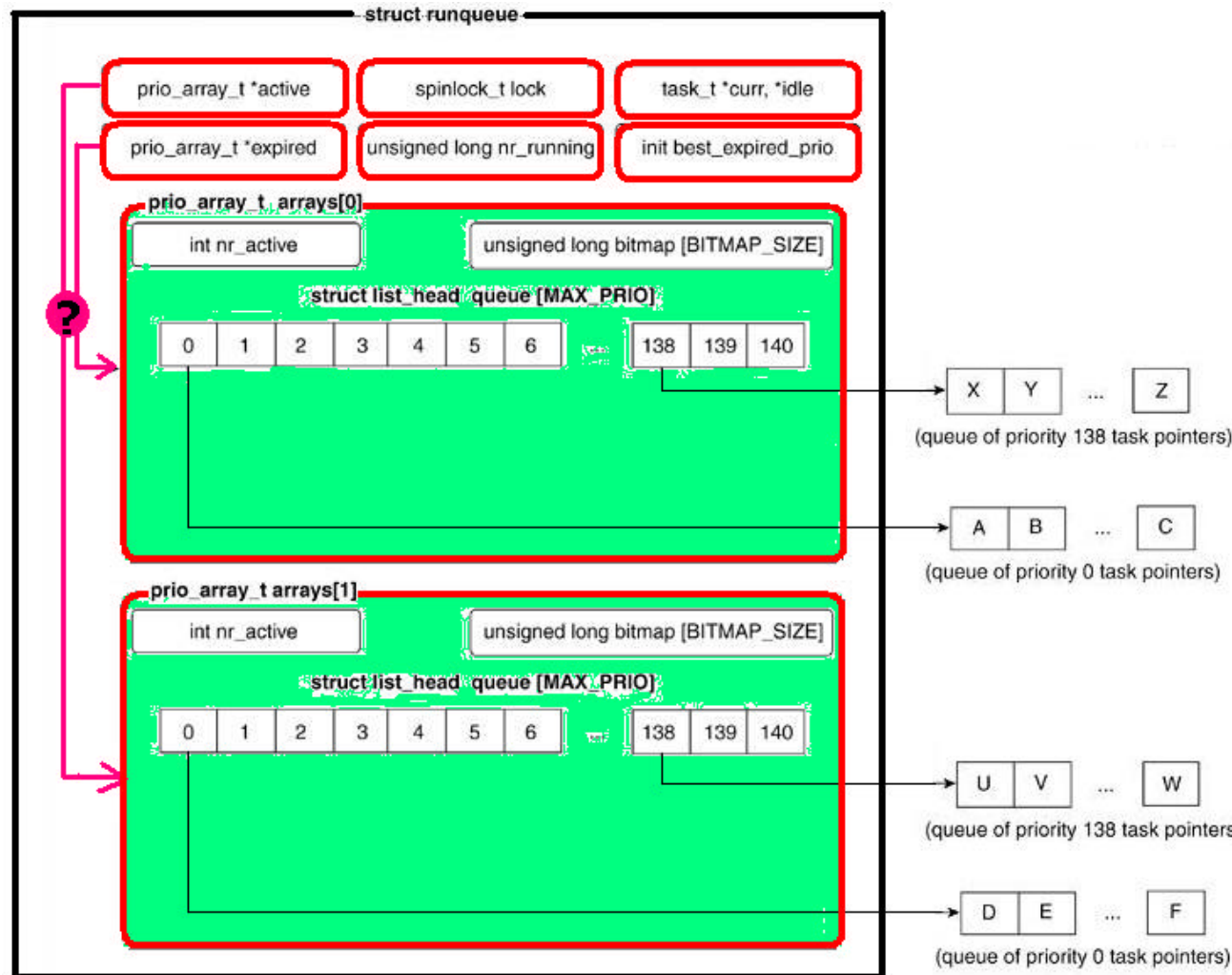
(queue of priority 0 task pointers)

?

```

05 struct rq_t
06     spinlock_t lock;
07     unsigned long nr_running;
08     unsigned long nr_uninterruptible;
09     struct task_struct *curr, *idle;
10     struct prio_array *active, *expired, arrays[2];
11     int best_expired_prio;
12     // . . .
13 };

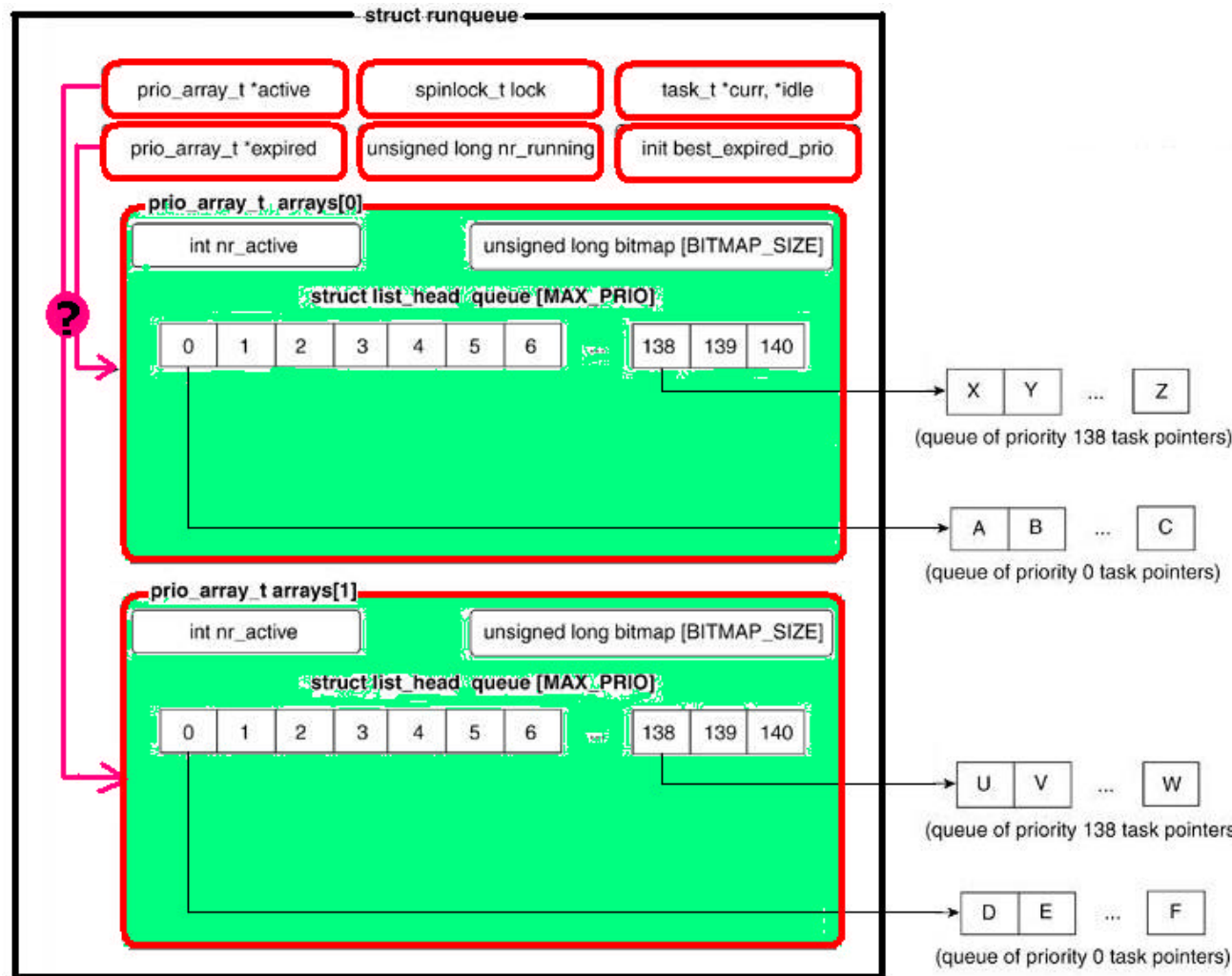
```



```

92 struct prio_array {
93     unsigned int nr_active;
94     unsigned long bitmap[BITS_TO_LONGS(MAX_PRIO+1)];
95     struct list_head queue[MAX_PRIO];
96 };

```



nr_active

– 'queue[]'

bitmap

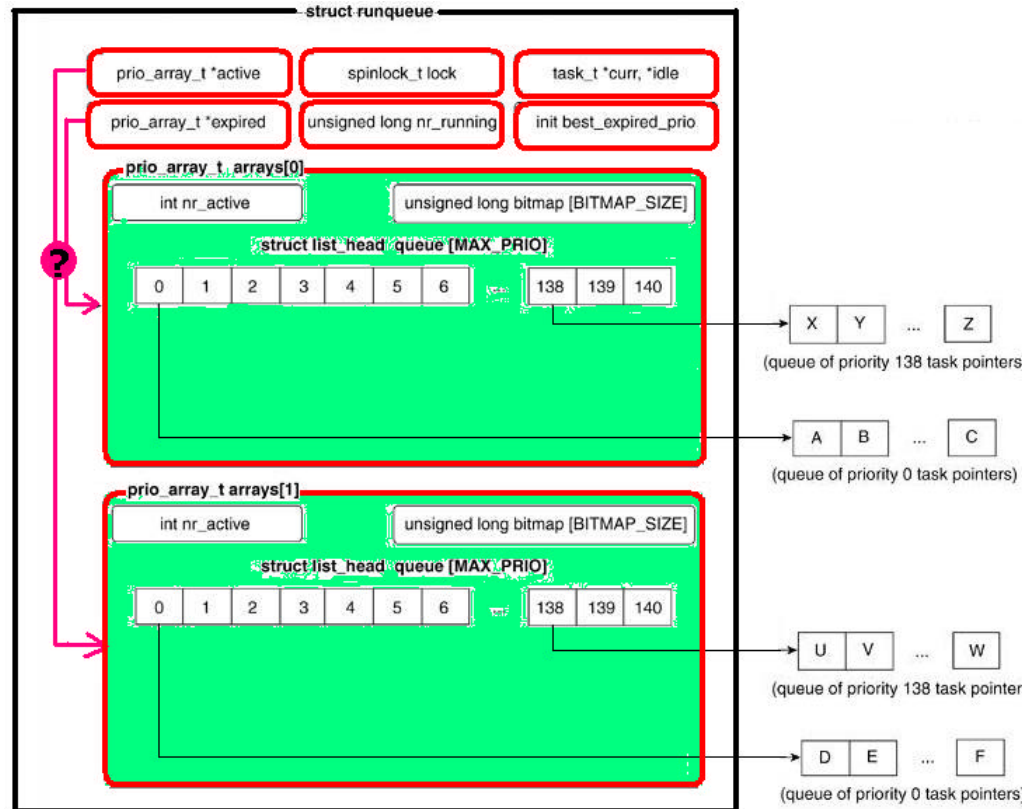
– queue[i] 가 , bitmap

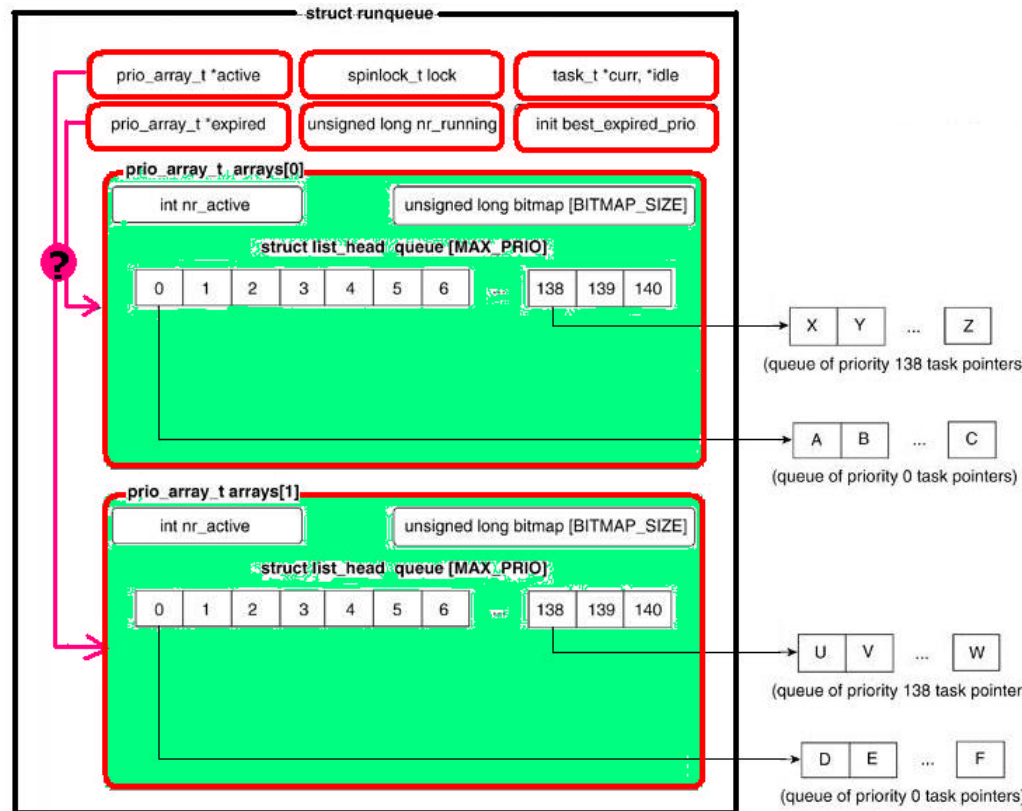
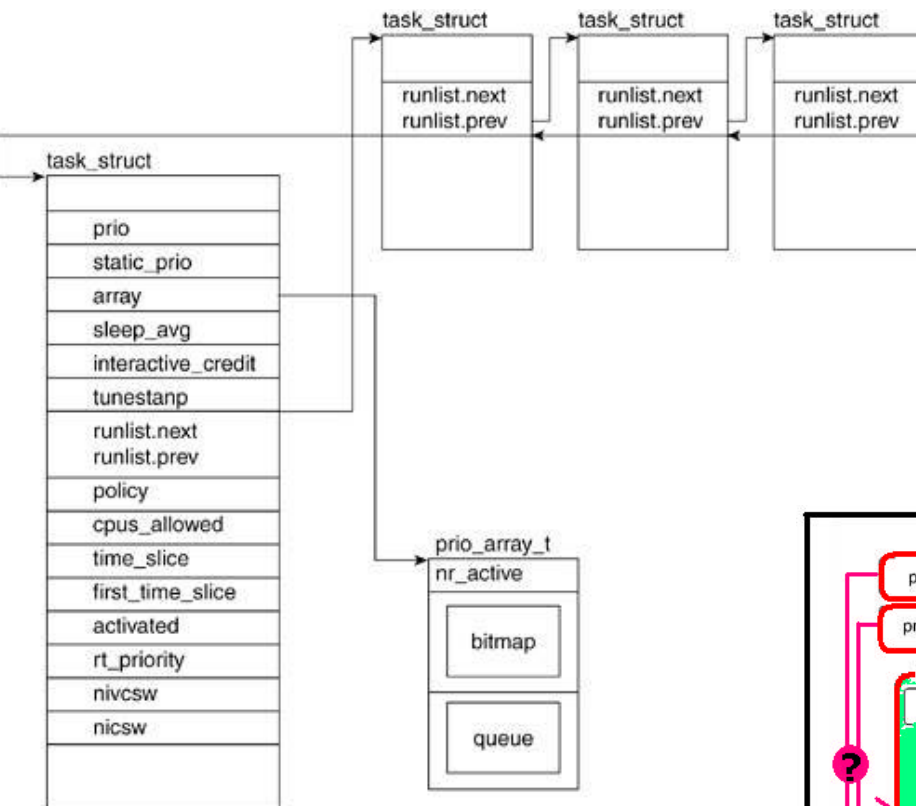
i 가 1

queue[]

– queue[i] , prio i

– See enqueue_task() or
requeue_task()





3.6. Keeping Track of Processes

- runqueue
 - runqueue 가?
 - static DEFINE_PER_CPU(strcut rq, runqueues);
// in kernel/sched.c
 - CPU runqueue?
 - rq = this_rq();

3.6. Keeping Track of Processes

- Scheduler (schedule()) 가
 - if ((current->state == TASK_INTERRUPTIBLE)
&& signal_pending(current))
current->state = TASK_RUNNING;
 - if (current->state != TASK_RUNNING)
deactivate_task(current, rq); // runqueue
 - (next 가 PCB 가)
 - context switch by...
prev = context_switch(rq, prev, next);
↳ switch_mm(); switch_to();

3.6. Keeping Track of Processes

- Scheduler (schedule()) 가
 - if ((current->state == TASK_INTERRUPTIBLE) && signal_pending(current))
current->state = TASK_RUNNING;
 - if (current->state != TASK_RUNNING)
deactivate_task(current, rq); // runqueue
 - (next 가 PCB 가)
 - context switch by...
prev = context_switch(rq, prev, next);
↳ switch_mm(); switch_to();

,
active
array가
,
active
expired

3.6. Keeping Track of Processes

- Scheduler (schedule())

- if ((current->state == TASK_RUNNING) && signal_pending(current)) {
 current->state = TASK_RUNNING;
 if (current->state != TASK_RUNNING) deactivate_task(current);
 if (next == NULL) {
 CPU가 PC를 가 (prev) switch_to();
 context switch by...
 prev = context_switch(rq, prev, next);
 ↳ switch_mm(); switch_to();
 }

active array가 active expired

switch_to() switching context .
가 (prev) switch_to() CPU

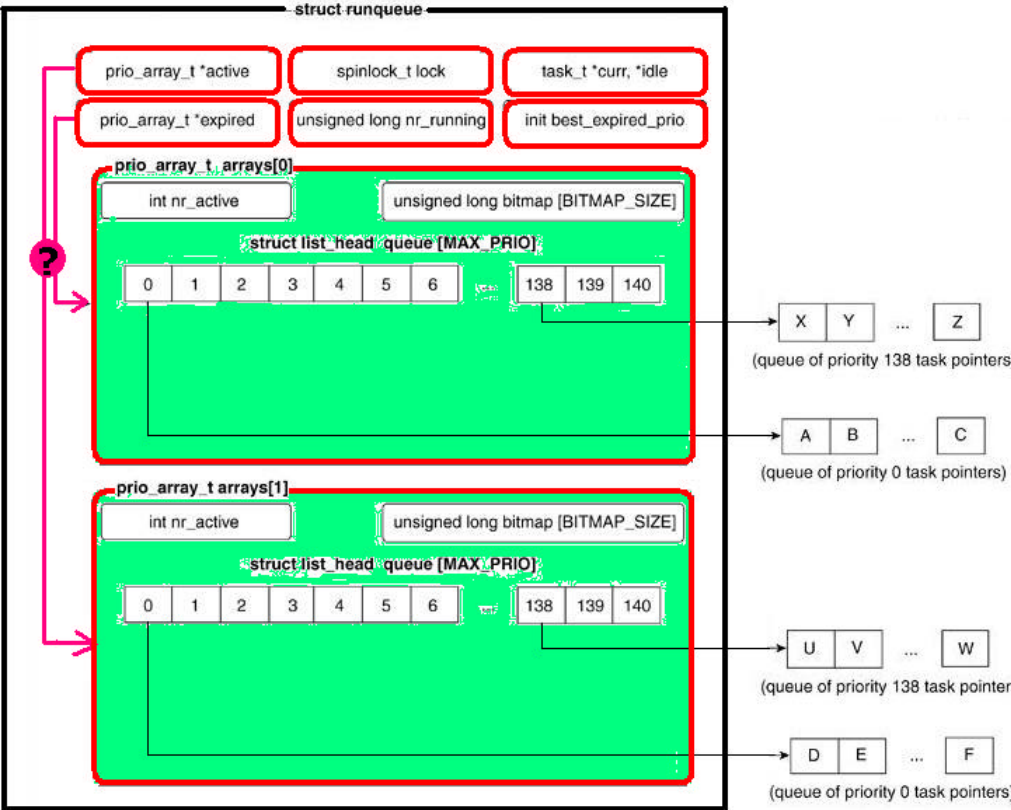
3.6. Keeping Track of Processes

- time_slice (1/2)
 - task_struct time_slice
 - , “ 가
CPU 가”
 - , 가 CPU

3.6. Keeping Track of Processes

- time_slice (2/2)
 - , ,
current->time_slice .
 - timer_interrupt() do_timer_interrupt_hook()
update_process_times() **schedule_tick()** .
 - current->time_slice 가 0 ,
'active' , 'expired' .
(schedule_tick())
 - , interactive , active

- active
 - 가 expired 가
 - ?
 - time_slice가
 - schedule()
 - sched_yield()
 - yield()



3.6. Keeping Track of Processes

- Process ,

do_fork()

└→ copy_process()

└┐ sched_fork(): **task_struct**

└→ wake_up_new_task(): **runqueue**

3.6. Keeping Track of Processes

- Process ,

do_fork()

└→ copy_process()

└┐ sched_fork(): **task_struct**

└→ wake_up_new_task(): **runqueue**

:

wake_up_forked_process()

3.6. Keeping Track of Processes

- Process ,

do_fork()

└→ copy_process()

└┐ sched_fork(): **task_struct**

└→ wake_up_new_task(): **runqueue**

Why separated?

-

3.6. Keeping Track of Processes

- Process ,

do_fork()

→ copy_process()

└─→ sched_fork(): **task_struct**

└─→ wake_up_new_task(): **runqueue**

Let's investigate this first!

- sched_fork() (in kernel/sched.c)
 - p->state = TASK_RUNNING;
 - run_list array
 - (runqueue ,)
 - time_slice
 - p->time_slice = (current->time_slice + 1) >> 1;
 - current->time_slice >>= 1;
 - (가 fork() CPU)
 - p->first_time_slice = 1;
 - time_slice
 - time_slice가 0 schedule_tick()

- sched_fork

- p->state =

- run_list

- (runque

- time_slice

- p->time_slice = (current->time_slice + 1)

- current->time_slice >>= 1;

- (가

fork()
)

CPU

- p->first_time_slice = 1;

- time_slice

- time_slice가 0

schedule_tick()

(1)

' active '

'expired'

(2) prio

(3)

가

3.6. Keeping Track of Processes

- Process ,

do_fork()

→ copy_process()

└→ sched_fork(): task struct

→ wake_up_new_task(): runqueue

now look into this function

- `wake_up_new_task()` (in `kernel/sched.c`) (1/2)
 - `runqueue` lock .
 - `sched_fork()` state가 `TASK_RUNNING` `BUG();`
 - `sleep_avg` (sleep average) .
 - sleep 가 .
 - 가 interactive .
 - `effective_prio()` ().
 - `schedule_tick()` active () .

- `wake_up_new_task()` (in *kernel/sched.c*) (2/2)
 - `prio` (dynamic priority)
 - `effective_prio()` 가 `__normal_prio()`
 - `static_prio`
 - `static_prio` `nice`
 - `sleep_avg`
 - `runqueue`
 - (list_add_tail()) , (?)
 - `task_struct` array
 - 가 `__activate_task()` `runqueue`
 - `runqueue` lock

Contents

3.0. Intro.

3.1. Introducing Our Program

3.2. Process Descriptor

3.3. Process Creation: `fork()`, `vfork()`, & `clone()`

3.4. Process Lifespan

3.5. Process Termination

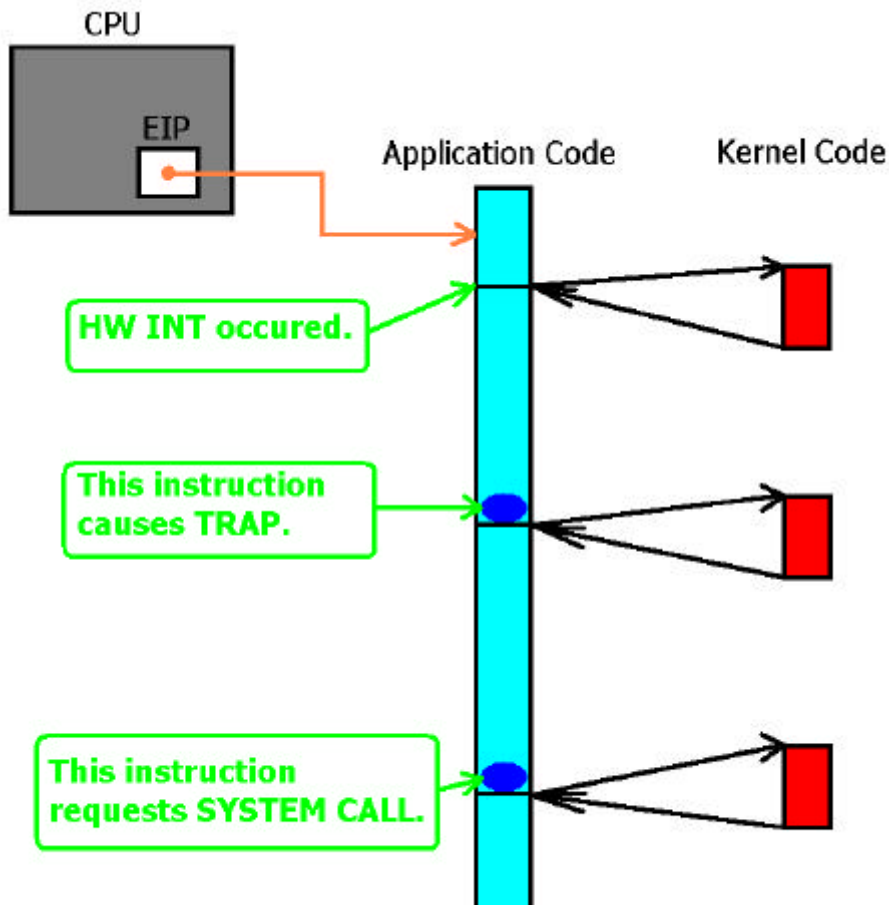
3.6. Keeping Track of Process: Basic Scheduler Construction

3.7. Wait Queues

3.8. Asynchronous Execution Flow

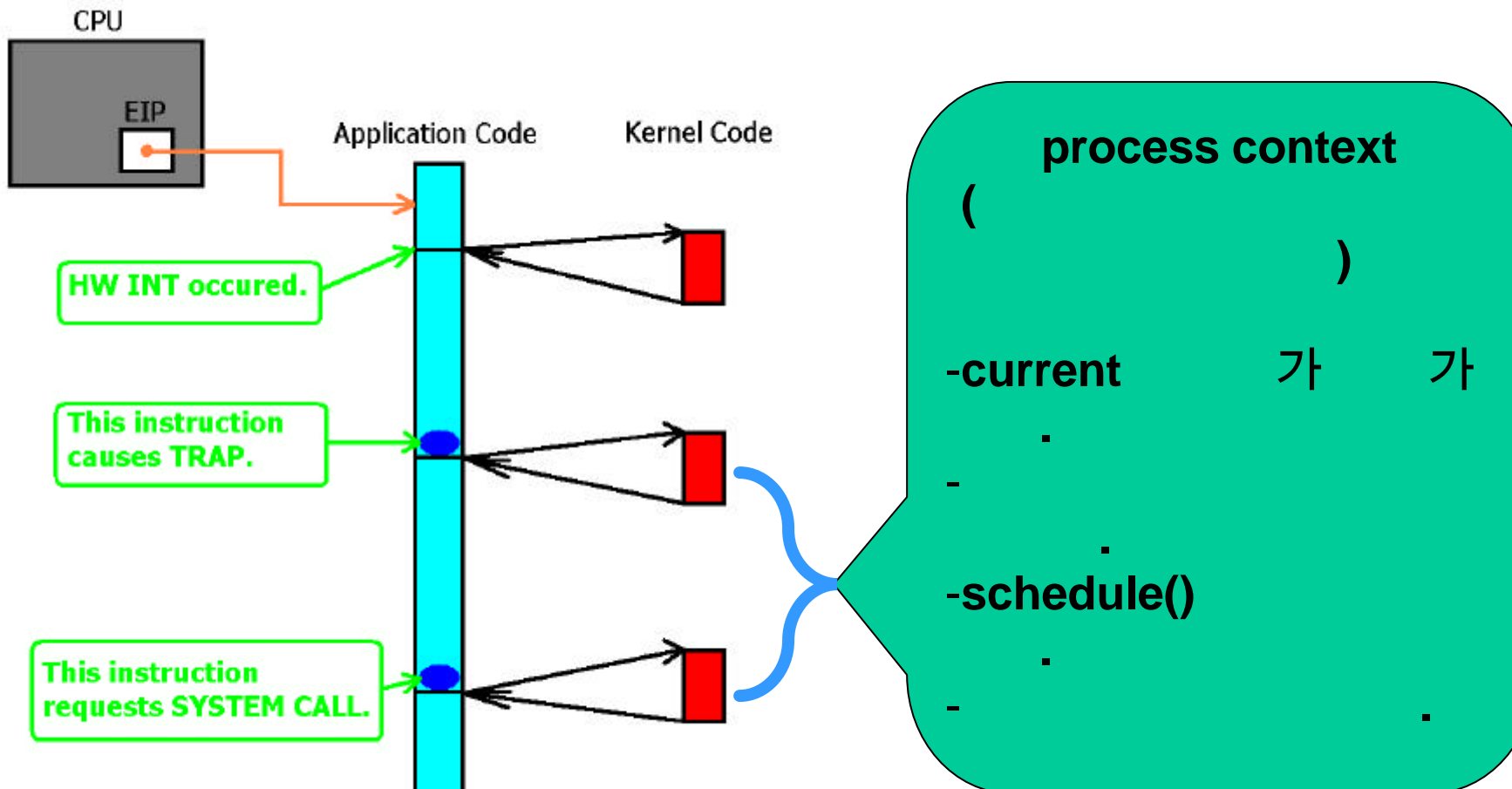
3.7. Wait Queues

- process context interrupt context



3.7. Wait Queues

- process context interrupt context



3.7. Wait Queues

- “ ”?
 - state TASK_(UN)INTERRUPTIBLE
 - runqueue .

3.7. Wait Queues

- exception
process context
- (current) ?
 - without a wait queue
 - using a wait queue
 - manually
 - `*sleep_on*()`: *obsolete & racy! so don't use them.*
 - `wait_event*()`
 - semaphore (`up()` and `down()`)
 - `wait_for_completion*()`

3.7. Wait Queues

- `pthread_mutex_t` exception
 - `pthread_mutexattr_t` process context
- `pthread_cond_t` (current) ?
 - without a wait queue
 - using a wait queue
 - manually
 - `*sleep_on*(): obsolete & racy! so don't use them.`
 - `wait_event*()`
 - semaphore (`up()` and `down()`)
 - `wait_for_completion*()`

- without a wait queue
(: TASK_RUNNING to TASK_INTERRUPTIBLE)

– 가 .

```
while (1) {
    if (resource_available) break;
    set_current_state(TASK_INTERRUPTIBLE);
    schedule();
}
set_current_state(TASK_RUNNING);
```

– event resource blocking I/O 가

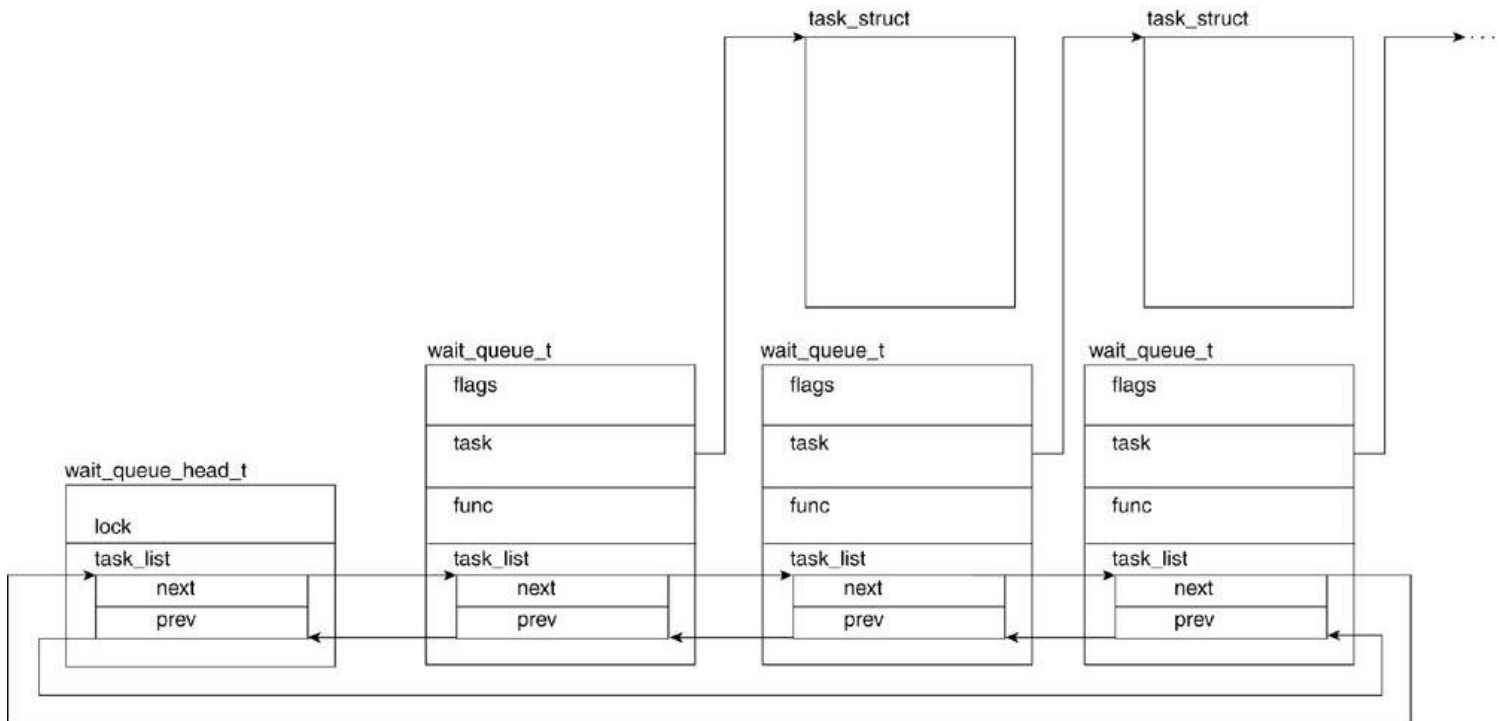
– resource가 signal ,
wake up (try to wake up()).

3.7. Wait Queues

- exception
process context
- (current) ?
 - without a wait queue
 - using a wait queue
 - manually
 - *sleep_on*(): *obsolete & racy! so don't use them.*
 - wait_event*()
 - semaphore (up() and down())
 - wait_for_completion*()

- Wait Queue
 - “~가

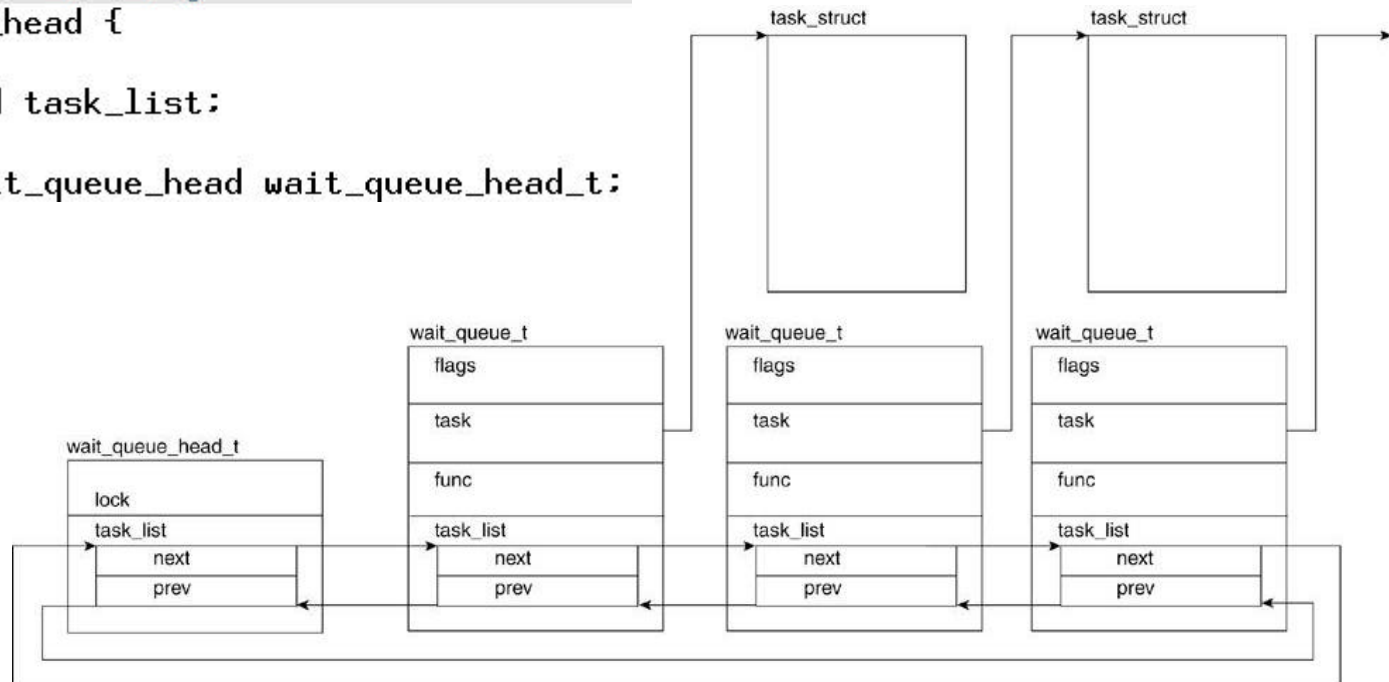
”



```

typedef struct __wait_queue wait_queue_t;
+--- 3 줄: typedef int (*wait_queue_func_t)(wait_q
struct __wait_queue {
    unsigned int flags;
#define WQ_FLAG_EXCLUSIVE    0x01
    void *private;
    wait_queue_func_t func;
    struct list_head task_list;
};
+--- 11 줄: struct wait_bit_key {-----
struct __wait_queue_head {
    spinlock_t lock;
    struct list_head task_list;
};
typedef struct __wait_queue_head wait_queue_head_t;

```



3.7. Wait Queues

- exception
process context
- (current) ?
 - without a wait queue
 - using a wait queue
 - manually
 - *sleep_on*(): *obsolete & racy! so don't use them.*
 - wait_event*()
 - semaphore (up() and down())
 - wait_for_completion*()

- manually
 - eg) *driver/char/rtc.c*

```

130 static DECLARE_WAIT_QUEUE_HEAD(rtc_wait);
131 +--196 줄 : #ifdef RTC_IRQ-----
327 static ssize_t rtc_read(struct file *file, char __user *buf,
328                          size_t count, loff_t *ppos)
329 {
330 +-- 3 줄 : #ifndef RTC_IRQ-----
333     DECLARE_WAITQUEUE(wait, current);
334 +-- 17 줄 : unsigned long data;-----
351     add_wait_queue(&rtc_wait, &wait);
352
353     do {
354 +-- 4 줄 : First make it right. Then make it fast. Putting this whole-----
358         __set_current_state(TASK_INTERRUPTIBLE);
359 +-- 6 줄 : spin_lock_irq (&rtc_lock);-----
365         if (data != 0)
366             break;
367
368         if (file->f_flags & O_NONBLOCK) {
369             retval = -EAGAIN;
370             goto out;
371         }
372         if (signal_pending(current)) {
373             retval = -ERESTARTSYS;
374             goto out;
375         }
376         schedule();
377     } while (1);
378
379     if (count == sizeof(unsigned int))
380         retval = put_user(data, (unsigned int __user *)buf) ?: sizeof(int);
381 +-- 4 줄 : else-----
385     out:
386         current->state = TASK_RUNNING;
387         remove_wait_queue(&rtc_wait, &wait);
388
389     return retval;
390 #endif
391 }

```

```

130 static DECLARE_WAIT_QUEUE_HEAD(rtc_wait);
131 +-- 190 줄: #ifdef RTC_ID0-----
327 static ssize_t rtc_read(struct file *file, char __user *buf,
328                          size_t count, loff_t *ppos)
329 {
330 +-- 0 줄: #ifdef RTC_IRQ-----
333 DECLARE_WAITQUEUE(wait, current);
334 +-- 17 줄: unsigned long data;-----
351 add_wait_queue(&rtc_wait, &wait);
352
353 do {
354 +-- 1 줄: First make it right. Then make it fast. Putting this whole-----
358 __set_current_state(TASK_INTERRUPTIBLE);
359 +-- 6 줄: spin_lock_irq(&rtc_lock);-----
365 if (data != 0)
366     break;
367
368 if (file->f_flags & O_NONBLOCK) {
369     retval = -EAGAIN;
370     goto out;
371 }
372 if (signal_pending(current)) {
373     retval = -ERESTARTSYS;
374     goto out;
375 }
376     schedule();
377 } while (1);
378
379 if (count == sizeof(unsigned int))
380     retval = put_user(data, (unsigned int __user *)buf) ?: sizeof(int);
381 +-- 4 줄: else-----
385 out:
386     current->state = TASK_RUNNING;
387     remove_wait_queue(&rtc_wait, &wait);
388
389     return retval;
390 #endif
391 }

```

```

232 irqreturn_t rtc_interrupt(int irq, void *dev_id, struct pt_regs *regs)
233 {
234 +-- 10 줄 : -----
244     if (is_hpet_enabled()) {
245 +-- 5 줄 : -----
250         rtc_irq_data |= (unsigned long)irq & 0xF0;
251     } else {
252         rtc_irq_data |= (CMOS_READ(RTC_INTR_FLAGS) & 0xF0);
253     }
254 +-- 11 줄 : if (rtc_status & RTC_TIMER_ON)-----
265     wake_up_interruptible(&rtc_wait);
266 +-- 3 줄 : kill_fasync (&rtc_async_queue, SIGIO, POLL_IN);-----
269     return IRQ_HANDLED;
270 }

```

```

232 irqreturn_t rtc_interrupt(int irq, void *dev_id, struct pt_regs *regs)
233 {
234 +-- 10 줄 : -----
244     if (is_hpet_enabled()) {
245 +-- 5 줄 : -----
250         rtc_irq_data |= (unsigned long)irq & 0xF0;
251     } else {
252         rtc_irq_data |= (CMOS_READ(RTC_INTR_FLAGS) & 0xF0);
253     }
254 +-- 11 줄 : if (rtc_status & RTC_TIMER_ON) -----
265     wake_up_interruptible(&rtc_wait);
266 +-- 3 줄 : kill_fasync(&rtc_async_queue, SIGIO, POLL_IN); -----
269     return IRQ_HANDLED;
270 }

```

```

21 void fastcall add_wait_queue(wait_queue_head_t *q, wait_queue_t *wait)
22 {
23     unsigned long flags;
24
25     wait->flags &= ~WQ_FLAG_EXCLUSIVE;
26     spin_lock_irqsave(&q->lock, flags);
27     __add_wait_queue(q, wait);
28     spin_unlock_irqrestore(&q->lock, flags);
29 }
30 +-- 2 줄 : EXPORT_SYMBOL(add_wait_queue);-----
32 void fastcall add_wait_queue_exclusive(wait_queue_head_t *q, wait_queue_t *wait)
33 {
34     unsigned long flags;
35
36     wait->flags |= WQ_FLAG_EXCLUSIVE;
37     spin_lock_irqsave(&q->lock, flags);
38     __add_wait_queue_tail(q, wait);
39     spin_unlock_irqrestore(&q->lock, flags);
40 }
41 +-- 2 줄 : EXPORT_SYMBOL(add_wait_queue_exclusive);-----

```

What does “EXCLUSIVE” mean??


```

146 #define wake_up(x)          __wake_up(x, TASK_UNINTERRUPTIBLE | TASK_INTERRUPTIBLE, 1, NULL)
147 #define wake_up_nr(x, nr)   __wake_up(x, TASK_UNINTERRUPTIBLE | TASK_INTERRUPTIBLE, nr, NULL)
148 #define wake_up_all(x)      __wake_up(x, TASK_UNINTERRUPTIBLE | TASK_INTERRUPTIBLE, 0, NULL)
149 #define wake_up_interruptible(x) __wake_up(x, TASK_INTERRUPTIBLE, 1, NULL)
150 #define wake_up_interruptible_nr(x, nr) __wake_up(x, TASK_INTERRUPTIBLE, nr, NULL)
151 #define wake_up_interruptible_all(x) __wake_up(x, TASK_INTERRUPTIBLE, 0, NULL)
152 #define wake_up_locked(x)    __wake_up_locked((x), TASK_UNINTERRUPTIBLE | TASK_INTERRUPTIBLE)
153 #define wake_up_interruptible_sync(x) __wake_up_sync((x), TASK_INTERRUPTIBLE, 1)

```

```

3561 static void __wake_up_common(wait_queue_head_t *q, unsigned int mode,
3562                             int nr_exclusive, int sync, void *key)
3563 {
3564     struct list_head *tmp, *next;
3565
3566     list_for_each_safe(tmp, next, &q->task_list) {
3567         wait_queue_t *curr = list_entry(tmp, wait_queue_t, task_list);
3568         unsigned flags = curr->flags;
3569
3570         if (curr->func(curr, mode, sync, key) &&
3571             (flags & WQ_FLAG_EXCLUSIVE) && !--nr_exclusive)
3572             break;
3573     }
3574 }

```

```

3575 +-- 8 줄: *-----
3583 void fastcall __wake_up(wait_queue_head_t *q, unsigned int mode,
3584                         int nr_exclusive, void *key)
3585 {
3586     unsigned long flags;
3587
3588     spin_lock_irqsave(&q->lock, flags);
3589     __wake_up_common(q, mode, nr_exclusive, 0, key);
3590     spin_unlock_irqrestore(&q->lock, flags);
3591 }

```

```

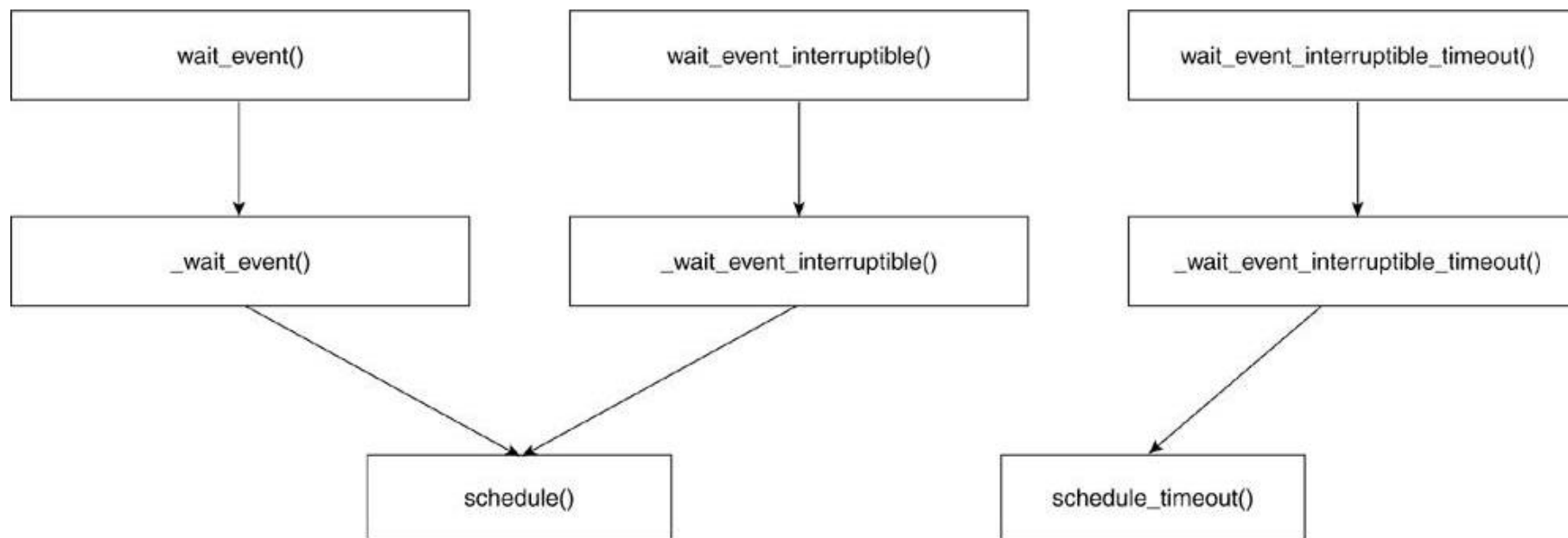
62 #define DECLARE_WAITQUEUE(name, tsk) \
63     .private      = tsk, \
64     .func         = default_wake_function, \
65     .task_list    = { NULL, NULL } }
66
67 #define DECLARE_WAITQUEUE(name, tsk) \
68     wait_queue_t name = __WAITQUEUE_INITIALIZER(name, tsk)
69
70 #define __WAIT_QUEUE_HEAD_INITIALIZER(name) { \
71     .lock        = __SPIN_LOCK_UNLOCKED(name.lock), \
72     .task_list   = { &(name).task_list, &(name).task_list } }
73
74 #define DECLARE_WAIT_QUEUE_HEAD(name) \
75     wait_queue_head_t name = __WAIT_QUEUE_HEAD_INITIALIZER(name)
76
1372 static int try_to_wake_up(struct task_struct *p, unsigned int state, int sync)
1373 {
1374     ++ 10 줄: int cpu, this_cpu, success = 0;-----
1384     rq = task_rq_lock(p, &flags);
1385     ++ 122 줄: old_state = p->state;-----
1507     activate_task(p, rq, cpu == this_cpu);
1508     ++ 14 줄: -----
1522 out_running:
1523     p->state = TASK_RUNNING;
1524 out:
1525     task_rq_unlock(rq, &flags);
1526
1527     return success;
1528 }
1529 ++ 2016 줄: int fastcall wake_up_process(struct task_struct *p)-----
3545 int default_wake_function(wait_queue_t *curr, unsigned mode, int sync,
3546                             void *key)
3547 {
3548     return try_to_wake_up(curr->private, mode, sync);
3549 }

```


3.7. Wait Queues

- exception
process context
- (current) ?
 - without a wait queue
 - using a wait queue
 - manually
 - `*sleep_on*()`: *obsolete & racy! so don't use them.*
 - **wait_event*()**
 - semaphore (`up()` and `down()`)
 - `wait_for_completion*()`

- `wait_event*()` (1/5)



- `wait_event*()` (2/5)
 - eg) *drivers/input/keyboard/sunkbd.c*
 - `sunkbd_connect()`
 - `init_waitqueue_head(&sunkbd->wait);`
 - `sunkbd_initialize()`
 - `wait_event_interruptible_timeout(sunkbd->wait, ...);`
 - `sunkbd_interrupt()`
 - `wake_up_interruptible(&sunkbd->wait);`

- wait_event*() (3/5)

```

267 #define __wait_event_interruptible_timeout(wq, condition, ret) \
268 do { \
269     DEFINE_WAIT(__wait); \
270 \
271     for (;;) { \
272         prepare_to_wait(&wq, &__wait, TASK_INTERRUPTIBLE); \
273         if (condition) \
274             break; \
275         if (!signal_pending(current)) { \
276             ret = schedule_timeout(ret); \
277             if (!ret) \
278                 break; \
279             continue; \
280         } \
281         ret = -ERESTARTSYS; \
282         break; \
283     } \
284     finish_wait(&wq, &__wait); \
285 } while (0)

```

- wait_event*() (3/5)

```
267 #define __wait_event_interruptible_timeout(wq, condition, ret) \
268 do { \
269     DEFINE_WAIT(__wait); \
270 \
271     for (;;) { \
272         prepare_to_wait(&wq, &__wait, TASK_INTERRUPTIBLE); \
273         if (condition) \
274             break; \
275         if (!signal_pending(current)) { \
276             ret = schedule_timeout(ret); \
277             if (!ret) \
278                 break; \
279             continue; \
280         } \
281         ret = -ERESTARTSYS; \
282         break; \
283     } \
284     finish_wait(&wq, &__wait); \
285 } while (0)
```

- wait_event*() (4/5)
 - simpler version in the textbook

```
1 #define __wait_event(wq, condition)
2 do {
3     wait_queue_t __wait;
4     init_waitqueue_entry(&__wait, current);
5
6     add_wait_queue(&wq, &__wait);
7     for (;;) {
8         set_current_state(TASK_UNINTERRUPTIBLE);
9         if (condition)
10             break;
11         schedule();
12     }
13     current->state = TASK_RUNNING;
14     remove_wait_queue(&wq, &__wait);
15 } while (0)
```

- `wait_event*()` (5/5)
 - See how “`wait_event_timeout()`” returns a value.

```
218 #define wait_event_timeout(wq, condition, timeout)
219 ({
220     long __ret = timeout;
221     if (!(condition))
222         __wait_event_timeout(wq, condition, __ret);
223     __ret;
224 })
```

Contents

3.0. Intro.

3.1. Introducing Our Program

3.2. Process Descriptor

3.3. Process Creation: `fork()`, `vfork()`, & `clone()`

3.4. Process Lifespan

3.5. Process Termination

3.6. Keeping Track of Process: Basic Scheduler Construction

3.7. Wait Queues

3.8. Asynchronous Execution Flow

3.8. Asynchronous Execution Flow

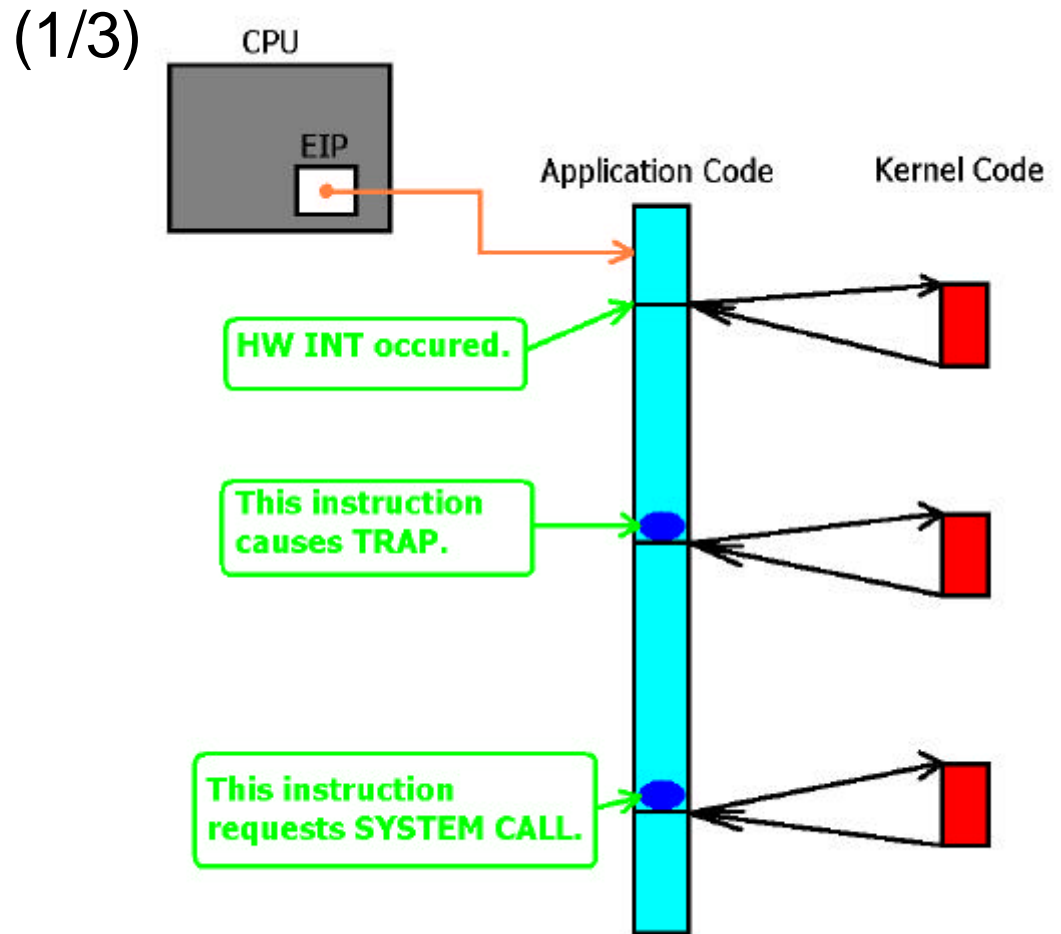
- ?
 - event가
 - CPU가
 - (, stack push jump .
 - x86 eflags, cs, eip, ss, esp)
- CPU 가 .
- , CPU
- , .

3.8. Asynchronous Execution Flow

- : x86
 - cs:eip
 -
 - ss:esp
 - top of stack

3.8. Asynchronous Execution Flow

-



3.8. Asynchronous Execution Flow

- (2/3)
 - exception or trap
 - . CPU
 - eg) divide by 0, page fault, invalid opcode, ...
 - “synchronous interrupt”
 - x86 int 0 ~19 (31)
 - hardware interrupt (‘ ’)
 - 가 CPU
 - “asynchronous interrupt”
 - x86 int 32 ~47 (48~127,129~255)
 - system call
 - user process가 OS
 - “software interrupt”
 - x86 int 128(=0x80)

3.8. Asynchronous Execution Flow

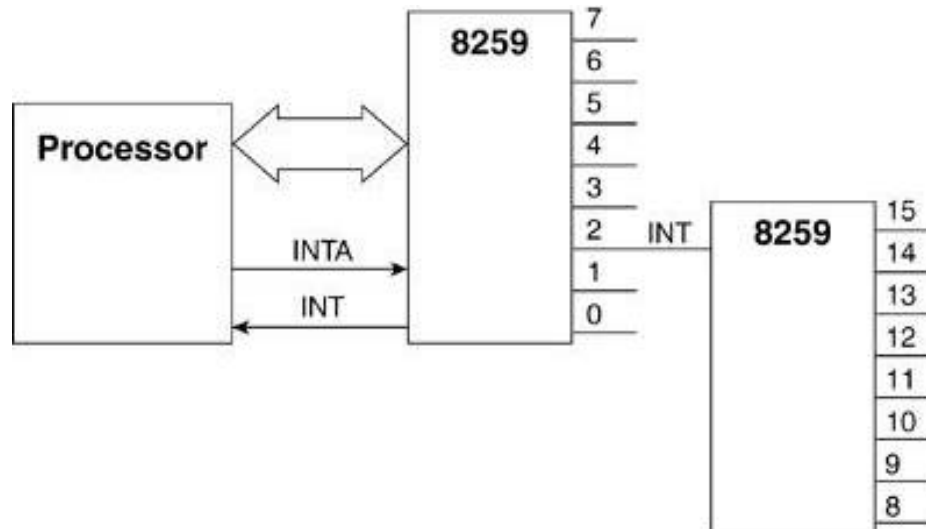
- (3/3)
* architecture , .

3.8. Asynchronous Execution Flow

* IRQ ?

-

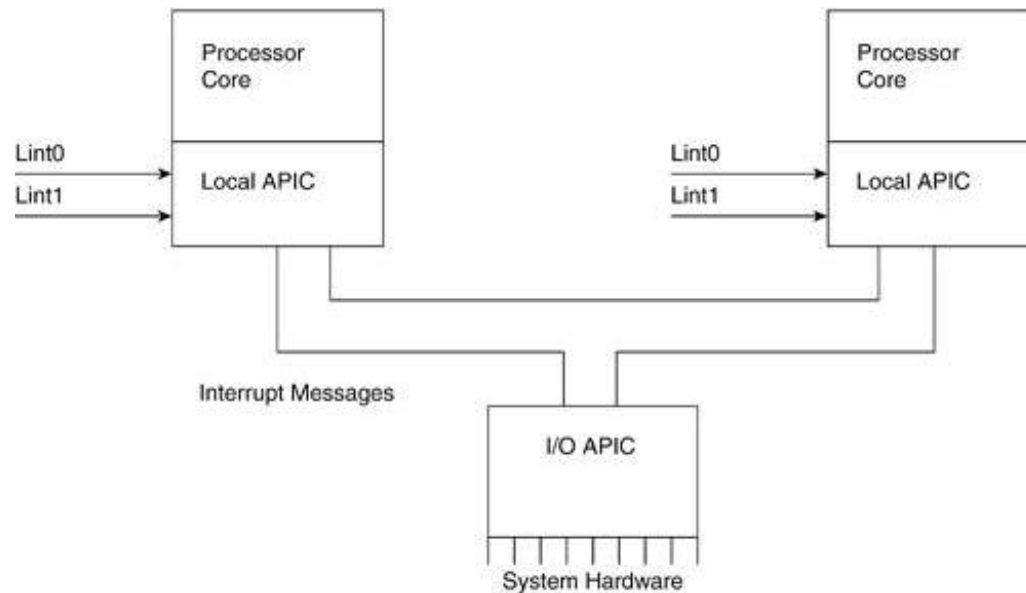
- x86 IRQ 0 int 32



3.8. Asynchronous Execution Flow

*

x86

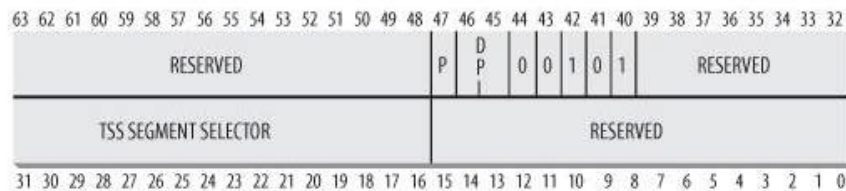


\$ cat /proc/interrupts

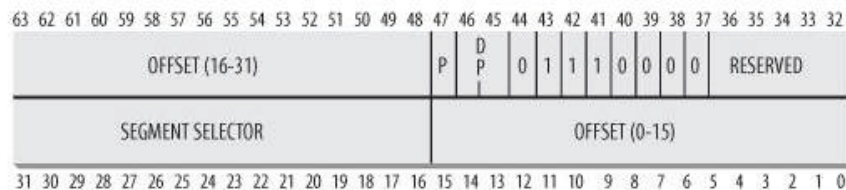
3.8. Asynchronous Execution Flow

- IDT (Interrupt Descriptor Table) ? (1/2)
 - 가 8bytes

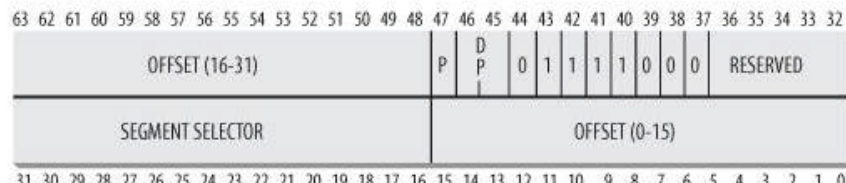
Task Gate Descriptor



Interrupt Gate Descriptor



Trap Gate Descriptor



3.8. Asynchronous Execution Flow

- IDT (Interrupt Descriptor Table) ? (2/2)

- CPU IDT n 가 ,

- , IDT 가
IDTR .

- *arch/i386/kernel/head.S* startup_32()
(lidt)

- idt_descr IDTR 가

- startup_32() setup_idt ,
IDT ignore_int .

3.8. Asynchronous Execution Flow

- Exception
- HW Interrupt
- System call

3.8. Asynchronous Execution Flow

- **Exception**
- HW Interrupt
- System call

- Exception (eg: divide error) (1/5)

—

- start_kernel()
 - trap_init()
 - » set_trap_gate(0, ÷_error);

IDT(Interrupt Descriptor Table)	0	entry
divide_error		.

- Exception (eg: divide error) (2/5)

- , CPU “divide error”
(0) 가

- CPU ,
 - eflags, cs, eip, ss, esp push
 - ss:esp
 - (x86)
 -
 - IDT[0]

- divide_error() 가

- Exception (eg: divide error) (3/5)

– : divide_error() (in *arch/i386/kernel/entry.S*)

```
ENTRY(divide_error)
```

```
    pushl $0                                # divide error, CPU가 0
```

```
    pushl $do_divide_error                 #
```

```
error_code:                               # : jump
```

```
    . . .
```

```
    movl ES(%esp), %edi                    # edi
```

```
    . . .
```

```
    call %edi                             #
```

```
    jmp ret_from_exception                 #
```

- Exception (eg: divide error) (4/5)
 - do_divide_error()
 - Can you find the definition?

arch/i386/kernel/traps.c

```
542 #define DO_VM86_ERROR_INFO(trapnr, signr, str, name, sicode, siaddr) \
543 fastcall void do_##name(struct pt_regs * regs, long error_code) \
544 { \
545     siginfo_t info: \
546     info.si_signo = signr: \
547     info.si_errno = 0: \
548     info.si_code = sicode: \
549     info.si_addr = (void __user *)siaddr: \
550     if (notify_die(DIE_TRAP, str, regs, error_code, trapnr, signr) \
551         == NOTIFY_STOP) \
552         return: \
553     do_trap(trapnr, signr, str, 1, regs, error_code, &info): \
554 }
555
556 DO_VM86_ERROR_INFO( 0, SIGFPE, "divide error", divide_error, FPE_INTDIV, regs->eip)
557 #ifndef CONFIG_KPROBES
558 DO_VM86_ERROR( 3, SIGTRAP, "int3", int3)
559 #endif
560 DO_VM86_ERROR( 4, SIGSEGV, "overflow", overflow)
561 DO_VM86_ERROR( 5, SIGSEGV, "bounds", bounds)
562 DO_ERROR_INFO( 6, SIGILL, "invalid opcode", invalid_op, ILL_ILLOPN, regs->eip)
563 DO_ERROR( 9, SIGFPE, "coprocessor segment overrun", coprocessor_segment_overrun)
564 DO_ERROR(10, SIGSEGV, "invalid TSS", invalid_TSS)
565 DO_ERROR(11, SIGBUS, "segment not present", segment_not_present)
566 DO_ERROR(12, SIGBUS, "stack segment", stack_segment)
567 DO_ERROR_INFO(17, SIGBUS, "alignment check", alignment_check, BUS_ADRALN, 0)
568 DO_ERROR_INFO(32, SIGSEGV, "iret exception", iret_error, ILL_BADSTK, 0)
```


- Exception (eg: divide error) (4/5)
 - do_divide_error()
 - Can you find the definition?
 - **How can we find it?**
 - :cs f s do_divide_error
 - :cs f g do_divide_error
 - :cs f e do_divide_error
 - grep -R . do_divide_error
 - see System.map and guess the location
 - make ***.i
 - gcc -E

- Exception (eg: divide error) (5/5)
 - do_trap():
 - current->thread.error_core = error_code;
 - // 가
 - // divide error 0
 - current->thread.trap_no = trapnr;
 - //
 - // divide error 0
 - force_sig(signr, current);
 - //
 - // (, current->pending->signal signr 1 set)
 - // divide error signr SIGFPE가

3.8. Asynchronous Execution Flow

- Exception
- **HW Interrupt**
- System call

- HW Interrupt (eg: timer interrupt) (1/11)

—

- start_kernel()
 - init_IRQ() (in *arch/i386/kernel/i8259.c*)
 - » set_intr_gate(vector, interrupt[I])

where is the array “interrupt[]” ??

- HW Interrupt (eg: timer interrupt) (1/11)

—

- start_kernel()
 - init_IRQ() (in arch/i386/kernel/i8259.c)
 - » set_intr_gate(vector, interrupt[I])

where is the array “interrupt[]” ??

```
include/linux/irq.h    include/asm-i386/hw_irq.h
extern void (*interrupt[NR_IRQS])(void); ???
```

- HW Interrupt (eg: timer interrupt) (2/11)
 - *arch/i386/kernel/entry.S* interrupt[]

```
ENTRY(interrupt)
.text
vector = 0
ENTRY(irq_entries_start)
.rept NR_IRQS          # .rept      .endr
    ALIGN
1:  pushl $(vector)    # irq
    jmp common_interrupt
.data
    .long 1b          #          NR_IRQS
.text
vector=vector+1
.endr
```

- HW Interrupt (eg: timer interrupt) (3/11)

- timer interrupt (irq = 0) 가 .
- CPU
 - eflags,cs,eip,ss,esp push
 - ss:esp
 - (x86)
 -
 - IDT[32]

- HW Interrupt (eg: timer interrupt) (4/11)

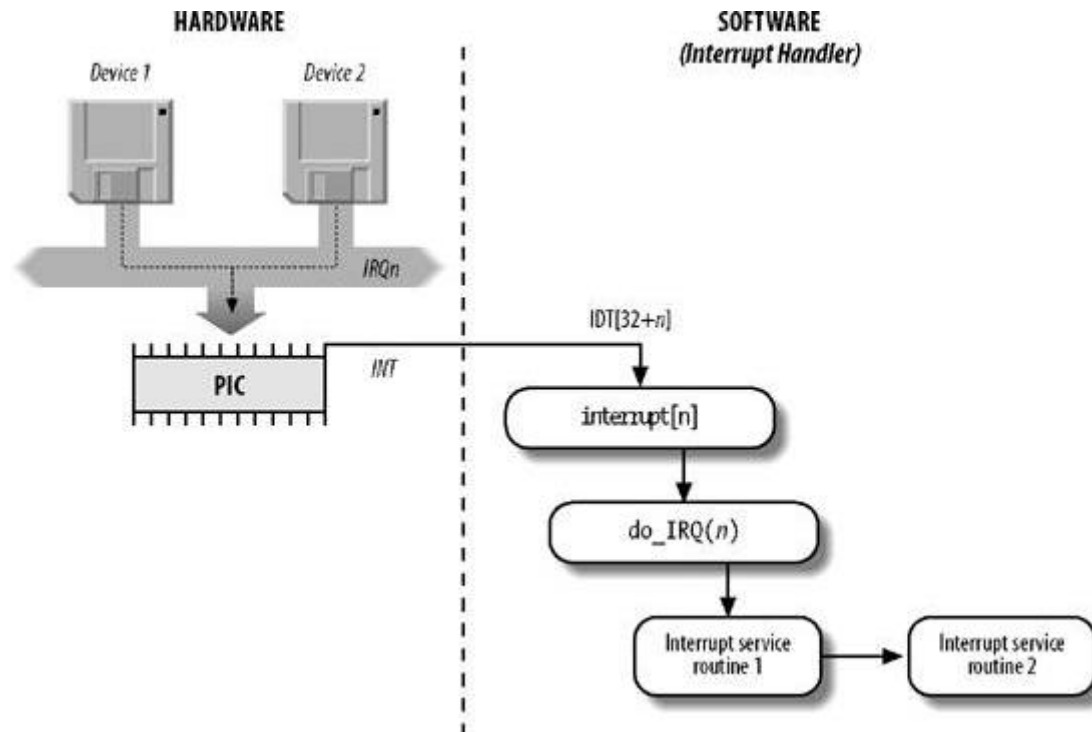
- IDT[32] , interrupt[0]
가 .

- pushl \$~0
 - jmp common_interrupt

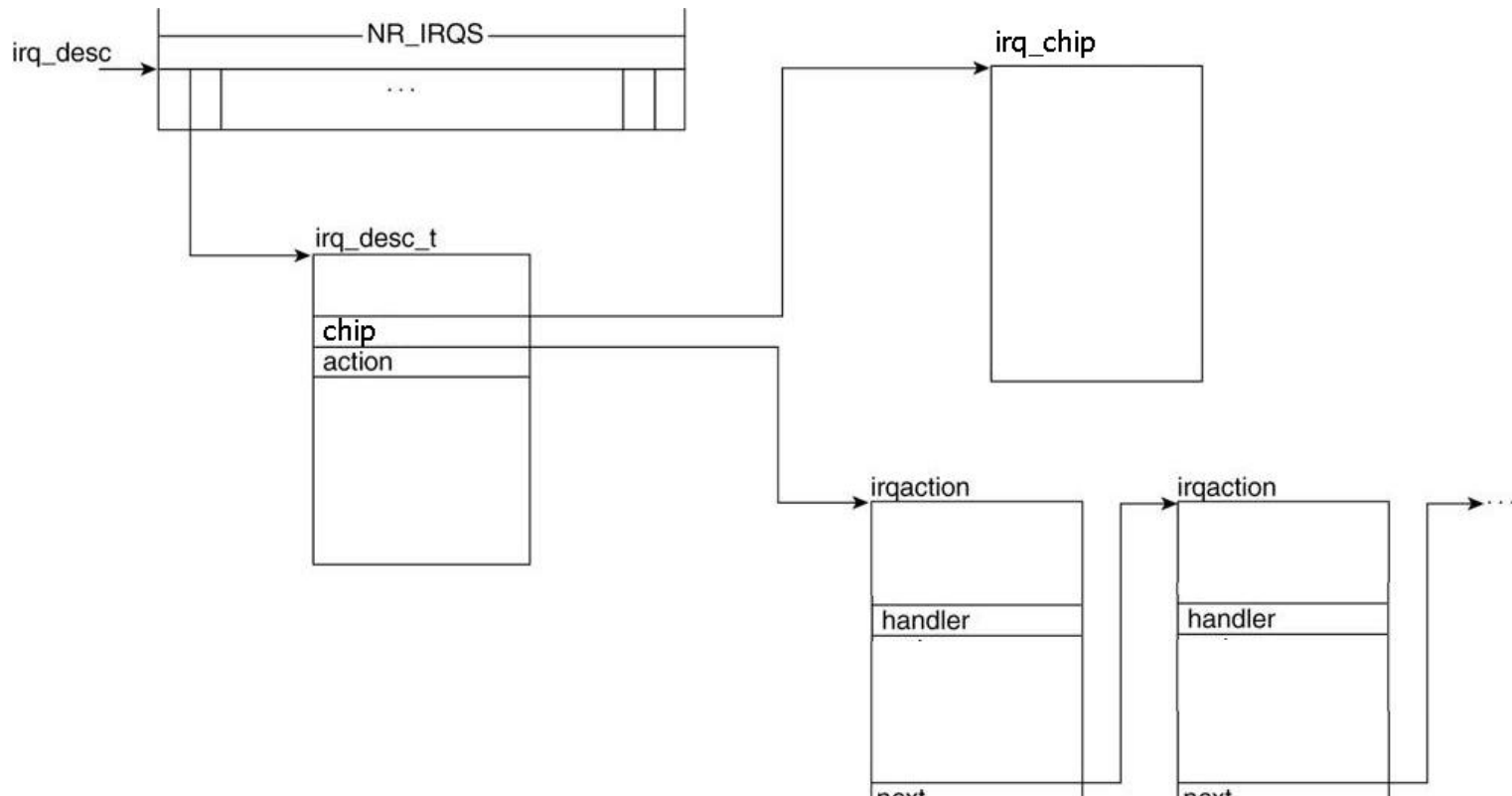
- common_interrupt:

- SAVE_ALL # push
 - movl %esp, %eax # top of stack eax
 - call do_IRQ
 - jmp ret_from_intr #

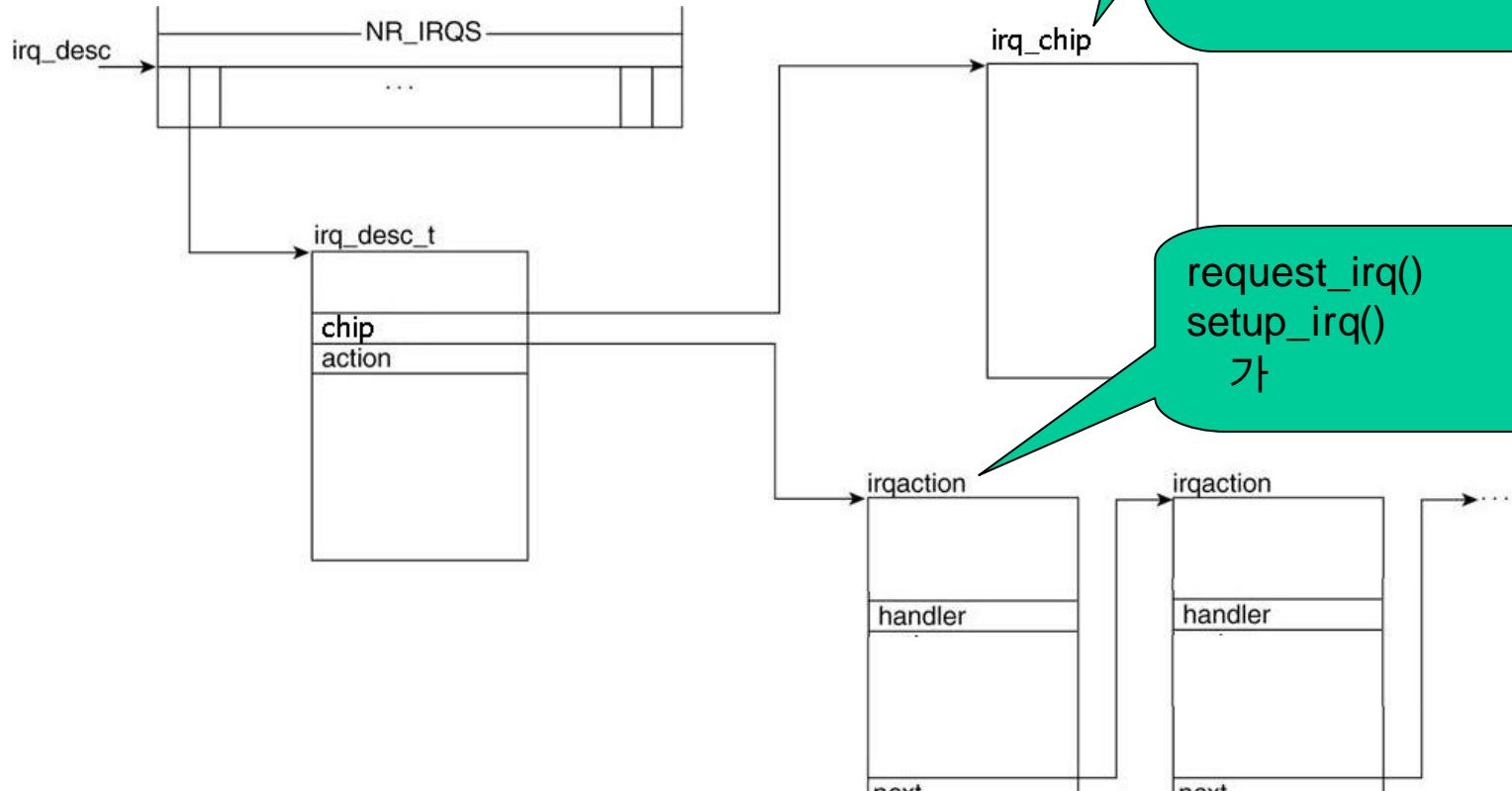
- HW Interrupt (eg: timer interrupt) (5/11)
 - call flow



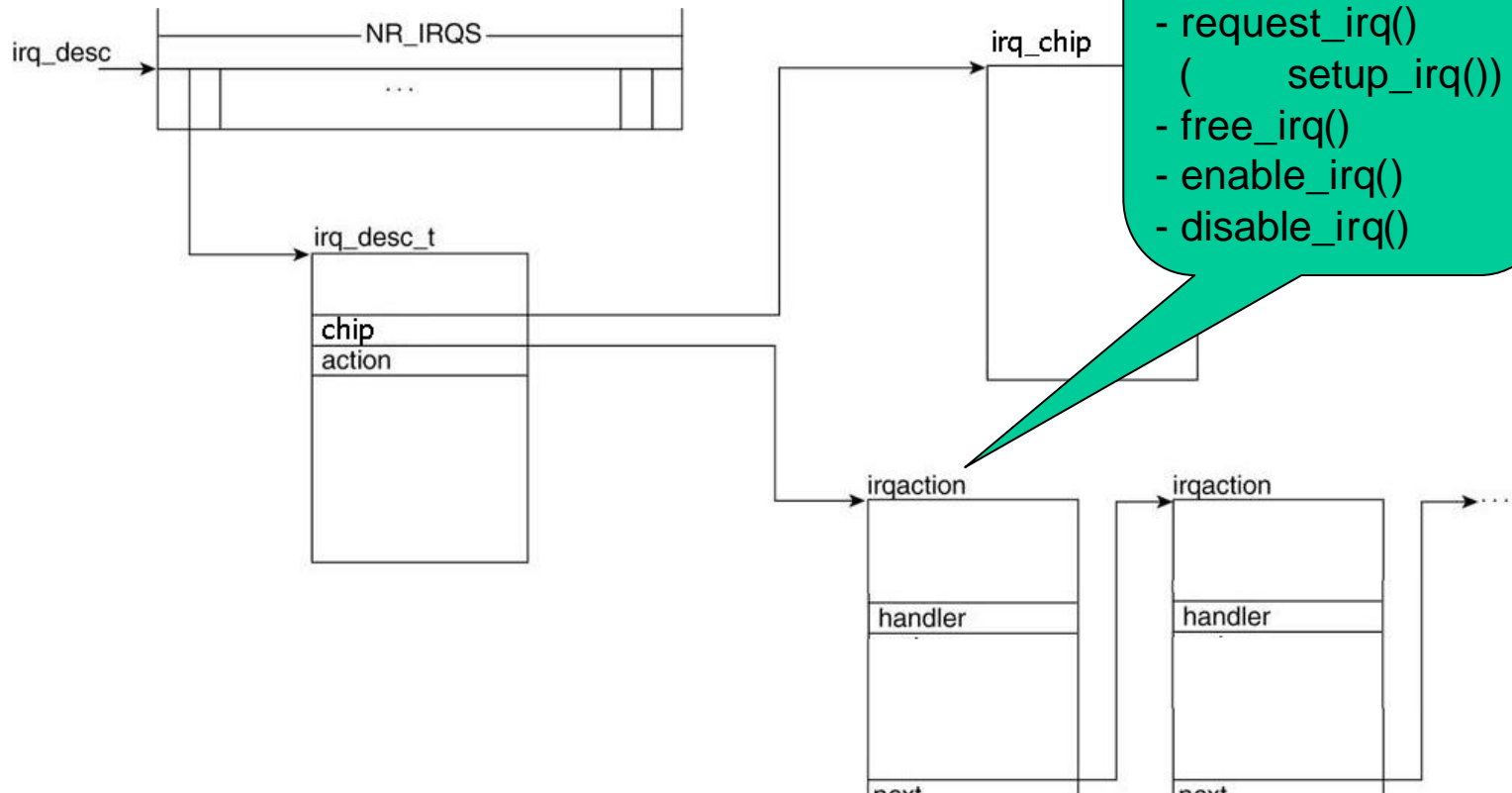
- HW Interrupt (eg: timer interrupt) (6/11)
 - do_IRQ()



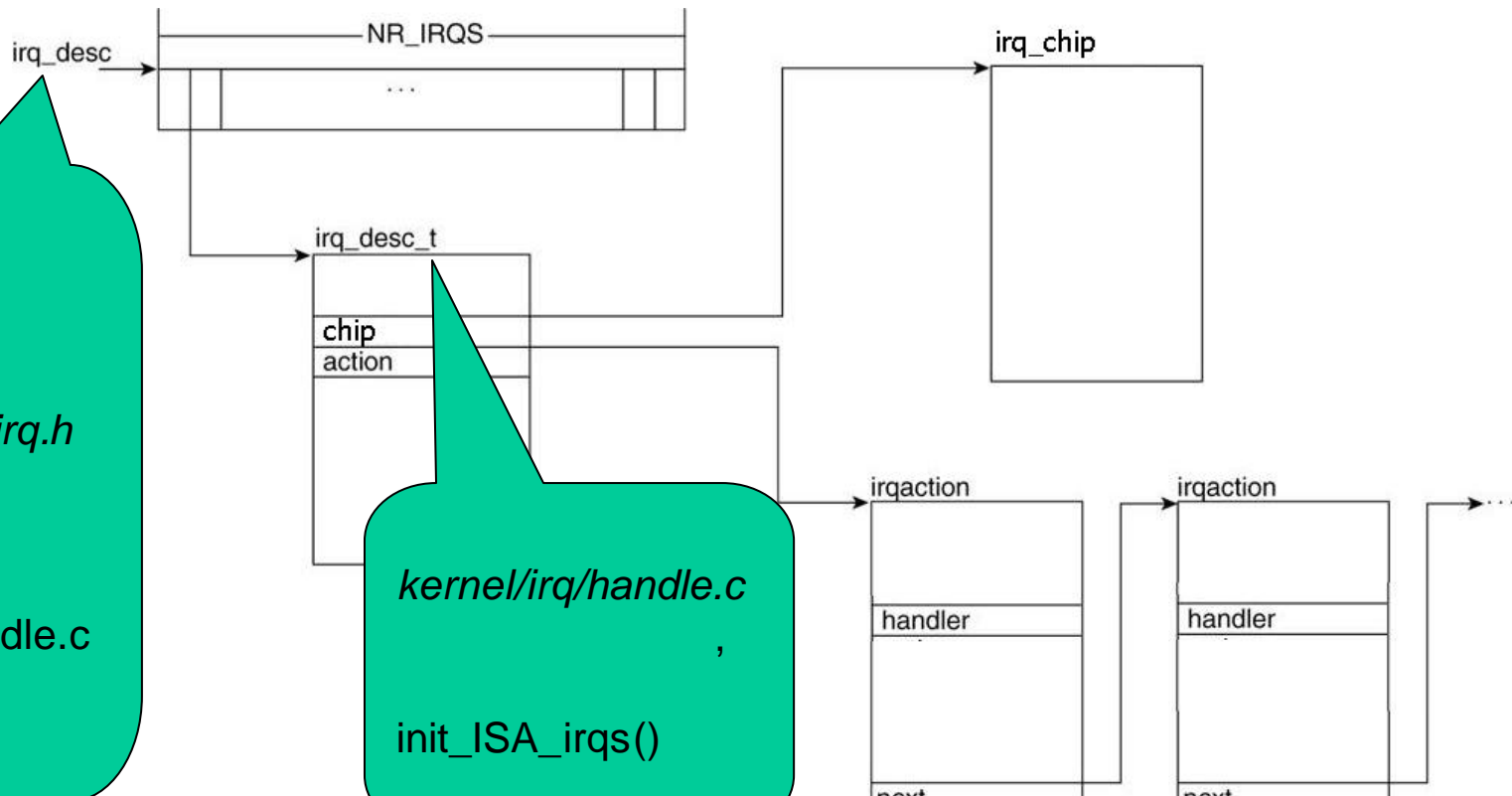
- HW Interrupt (eg: timer interrupt) (6/11)
 - do_IRQ()



- HW Interrupt (eg: timer interrupt) (6/11)
 - do_IRQ()



- HW Interrupt (eg: timer interrupt) (6/11)
 - do_IRQ()



irq_desc[]
가?
include/linux/irq.h
??
kernel/irq/handle.c
51

kernel/irq/handle.c
init_ISA_irqs()

- HW Interrupt (eg: timer interrupt) (7/11)
 - do_IRQ() (1/2)
 - prototype: **fastcall unsigned int do_IRQ(struct pt_regs *regs)**

- `regs` , `common_interrupt` `eax` top of stack .
- `regs` , CPU `SAVE_ALL` push
가 .
- `irq = ~regs->orig_eax;` , `pushl &~(vector)`
IRQ .

```

101 #define SAVE_ALL \
102     cld; \
103     pushl %es; \
104 +--- 2 줄: CFI_ADJUST_CFA_OFFSET 4;\--
106     pushl %ds; \
107 +--- 2 줄: CFI_ADJUST_CFA_OFFSET 4;\--
109     pushl %eax; \
110 +--- 2 줄: CFI_ADJUST_CFA_OFFSET 4;\--
112     pushl %ebp; \
113 +--- 2 줄: CFI_ADJUST_CFA_OFFSET 4;\--
115     pushl %edi; \
116 +--- 2 줄: CFI_ADJUST_CFA_OFFSET 4;\--
118     pushl %esi; \
119 +--- 2 줄: CFI_ADJUST_CFA_OFFSET 4;\--
121     pushl %edx; \
122 +--- 2 줄: CFI_ADJUST_CFA_OFFSET 4;\--
124     pushl %ecx; \
125 +--- 2 줄: CFI_ADJUST_CFA_OFFSET 4;\--
127     pushl %ebx; \
128 +--- 2 줄: CFI_ADJUST_CFA_OFFSET 4;\--
130     movl $(__USER_DS), %edx; \
131     movl %edx, %ds; \
132     movl %edx, %es;
133

```

```

21 #define FRAME_SIZE :
22
23 /* this struct defines the state of the
24    stack during a signal
25
26 struct pt_regs {
27     long ebx;
28     long ecx;
29     long edx;
30     long esi;
31     long edi;
32     long ebp;
33     long eax;
34     int xds;
35     int xes;
36     long orig_eax;
37     long eip;
38     int xcs;
39     long eflags;
40     long esp;
41     int xss;
42 };
43

```

- HW Interrupt (eg: timer interrupt) (7/11)

- do_IRQ() (2/2)

- irq = ~regs->orig_eax; // irq
 - irq_enter(); //
 - 가 (current
 - thread_info)
 - , thread_union (= thread_info +) 가
 - 8kB가 4kB ,
 - thread_info
 - ,
 - hardirq_ctx[CPU] .
 - __do_IRQ(); // all-in-one
 - irq_exit(); //
 - softirq .

- HW Interrupt (eg: timer interrupt) (8/11)

- `__do_IRQ()`

- `spin_lock (& (irq_desc[irq].lock));`
- `irq_desc[irq].chip->ack(irq);`
- `IRQ_PENDING` (ack CPU가)
- 가 goto out
 - `IRQ_DISABLED:` IRQ disable
 - `IRQ_INPROGRESS:` CPU가 .
 - `irq_desc[irq].action == NULL`
- `IRQ_PENDING 0 , IRQ_INPROGRES 1`
- `spin_unlock(& (irq_desc[irq].lock));`
- `handle_IRQ_event(); // action`
- `IRQ_PENDING 1 , CPU가 do_IRQ .`
- `irq_desc[irq].chip->end(irq);`

- HW Interrupt (eg: timer interrupt) (9/11)

- handle_IRQ_event()

- IRQF_DISABLED 가 ,
local_irq_enable_in_hard_irq() CPU
 - .
 - IRQ
 - : 2007 “SA_INTERRUPT” “IRQF_DISABLED”
 - action 가 handler
 - add_interrupt_randomness()
 - local_irq_disable() // CPU

- HW Interrupt (eg: timer interrupt) (10/11)

- ,
.

- (irq 0) action->handler
가?

- start_kernel()

- time_init()

- time_init_hook()

- » setup_irq(0, &irq0);

- // where, irq0 = {timer_interrupt, IRQF_DISABLED, ...}

- HW Interrupt (eg: timer interrupt) (11/11)
 - “Run me later!”
 - Interrupt handler .
 - ,
 -
 - “ ”
 - (bottom half)
 - “ ”
 - Softirq
 - Tasklet
 - Workqueue

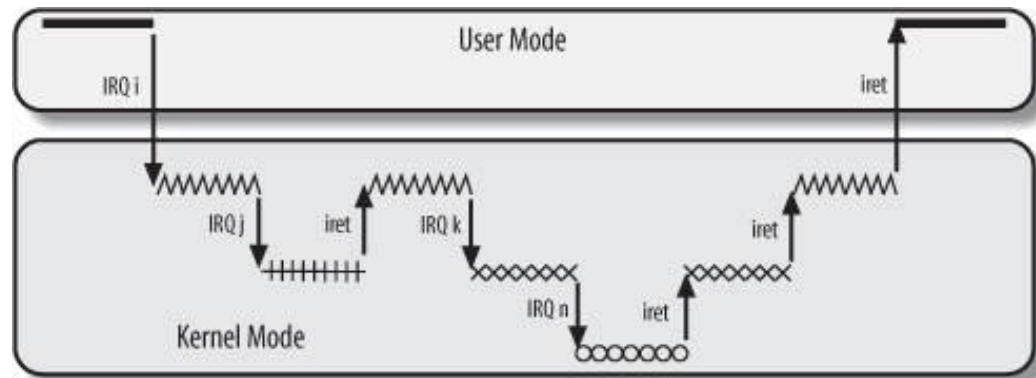
3.8. Asynchronous Execution Flow

- **Exception**
- **HW Interrupt**
- System call

- Exception & HW Interrupt (1/3)

– : Exception HW Interrupt ,
 (, exception page fault 가)

- HW interrupt handler ,
 HW interrupt handler exception handler
- Exception handler ,
 HW interrupt handler



- Exception & HW Interrupt (2/3)

- “Understanding the Linux Kernel” chap. 5

- : ,
 - : ,
 - : 가 ,

1. 가 , .

2. 가 ,

3. 가 ,

가 , .

4. , (“ ”)

- Exception & HW Interrupt (3/3)

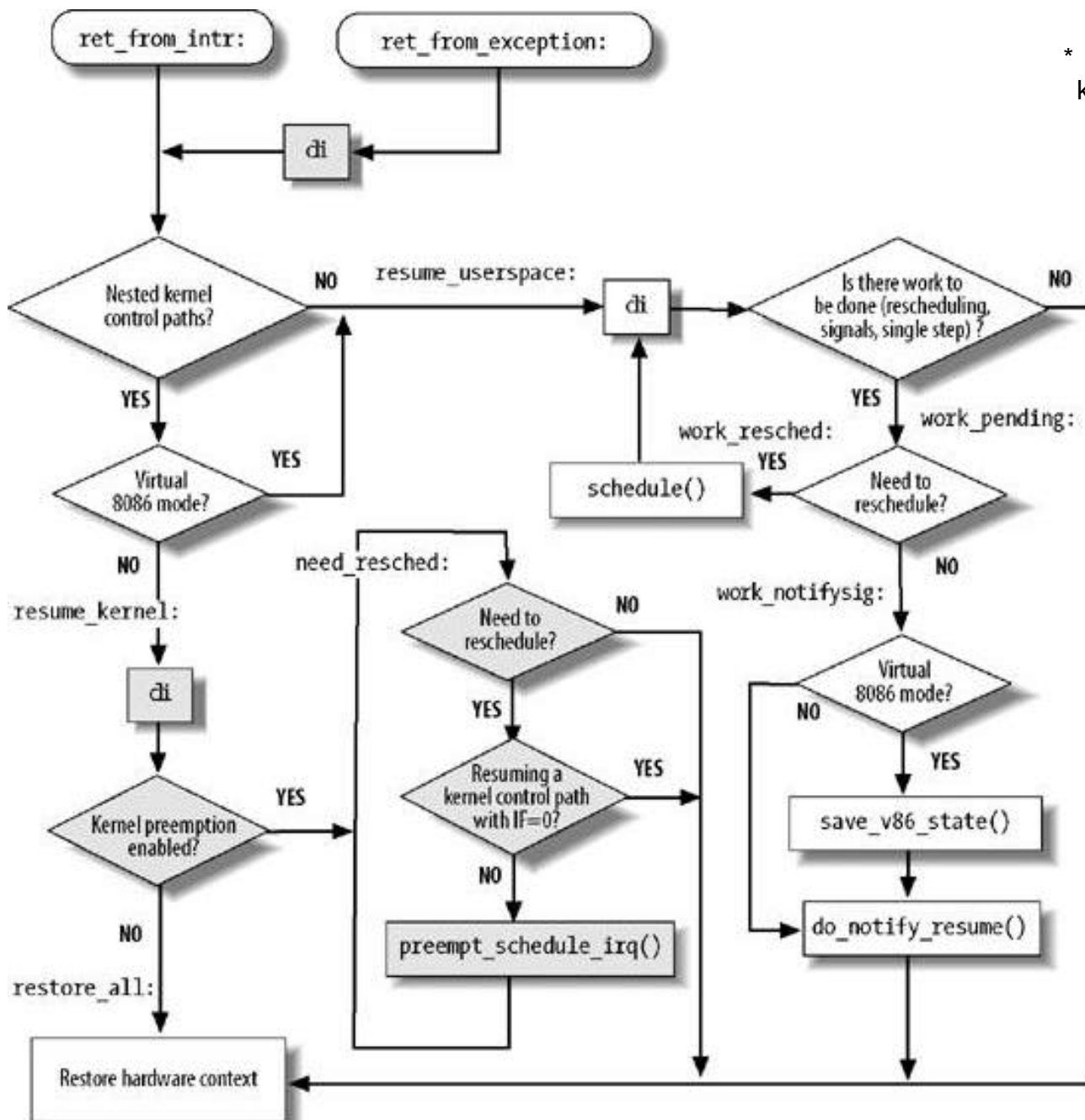
- Exception HW Interrupt

- ret_from_exception
 - ret_from_intr

가

*

kernel preemption



3.8. Asynchronous Execution Flow

- Exception
- HW Interrupt
- **System call**

- System call (1/4)

-

- start_kernel()

- trap_init()

- » set_system_gate(0x80, &system_call);

- IDT(Interrupt Descriptor Table) 0x80 entry
 - system_call .

- * system_call *arch/i386/kernel/entry.S* .

- System call (2/4)

- system_call (in *arch/i386/kernel/entry.S*)

- system call eax

- call sys_call_table[%eax] (in *arch/i386/kernel/syscall_table.S*)

- sys_***()

- » do_***() (sometimes)

- User mode header file

- */usr/include/bits/syscall.h*

- */usr/include/linux/unistd.h*

- System call (3/4)
 - copy_to_user(), copy_from_user()
 -
 - : _____
 - eg) sys_gettimeofday()

- System call (4/4)

- vsyscall

- Pentium 4
- int 0x80 sysenter sysexit
 ()
- user mode library calls __kernel_vsyscall()
- the instruction “sysenter” is executed
- the CPU jumps to “sysenter_entry” (in *arch/i386/kernel/entry.S*)
- reference
 - *arch/i386/kernel/vsyscall**
 - “Understanding Linux Kernel” 3rd ed., Chap. 10