Context Switching Accounting Mechanism

T. Castillo Girona

<toni.castillo@uoc.edu>

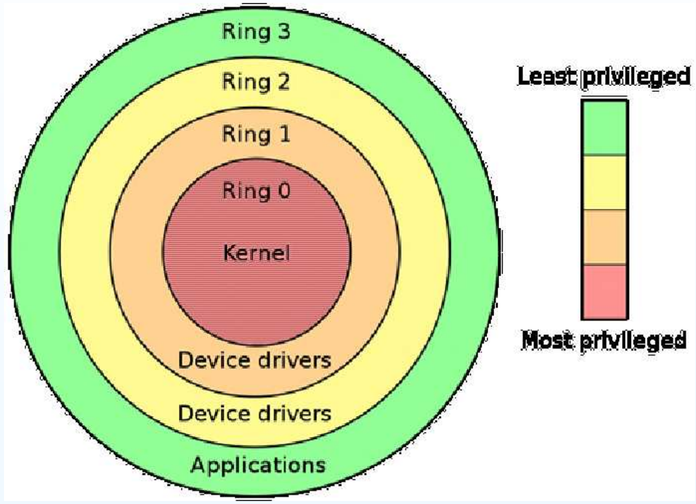
Miquel Angel Senar Rosell

Context Switching Basis

- Every single process p runs either in **Ring 0** or **Ring 3**.
  - **Ring 0**: *low-level or hardware tasks.*
  - **Ring 3**: *user-space tasks.*

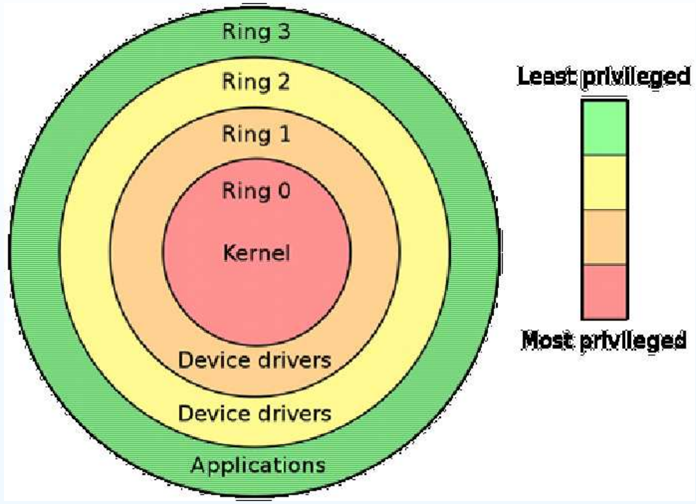
Context Switching Basis

- Every single process p runs either in **Ring 0** or **Ring 3**.
  - **Ring 0**: *low-level or hardware tasks.*
  - **Ring 3**: *user-space tasks.*



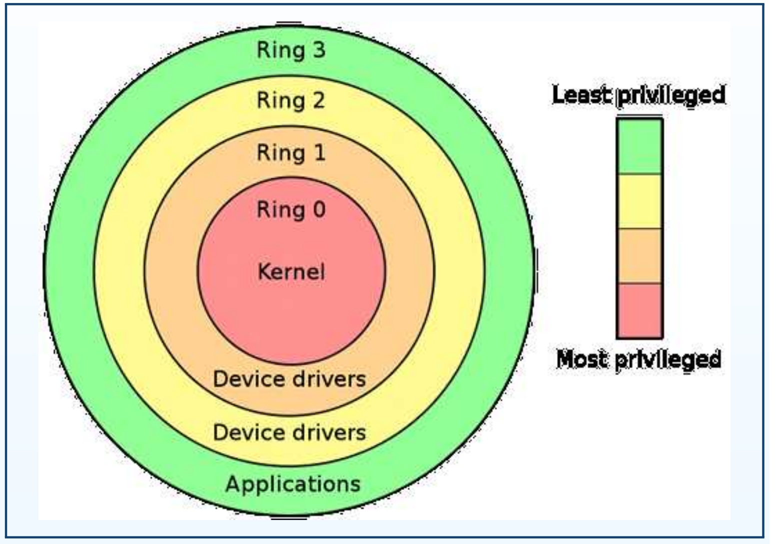
Context Switching Basis

- Every single process p runs either in **Ring 0** or **Ring 3**.
  - **Ring 0**: *low-level or hardware tasks.*
  - **Ring 3**: *user-space tasks.*



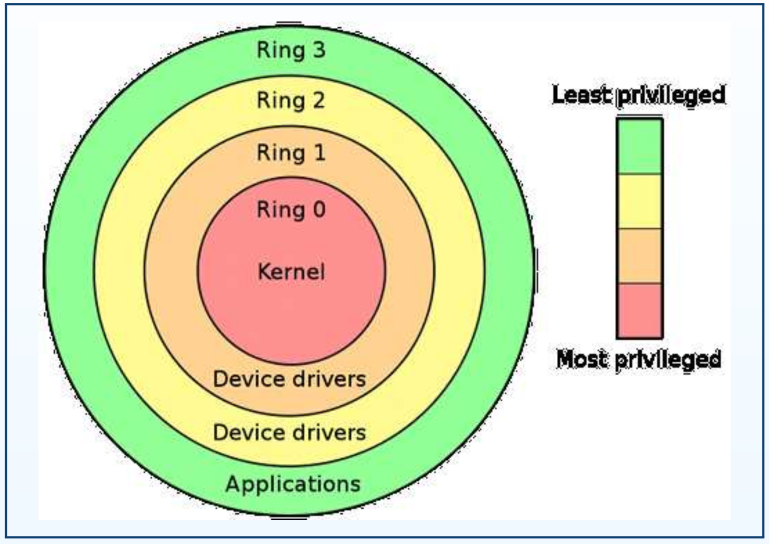
Context Switching Basis

- Every single process p runs either in **Ring 0** or **Ring 3**.
  - **Ring 0**: *low-level or hardware tasks.*
  - **Ring 3**: *user-space tasks.*
- p runs only for a measurable time t.



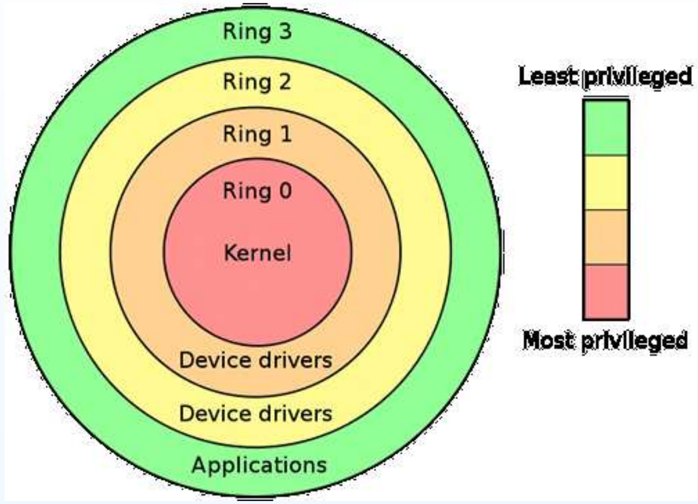
Context Switching Basis

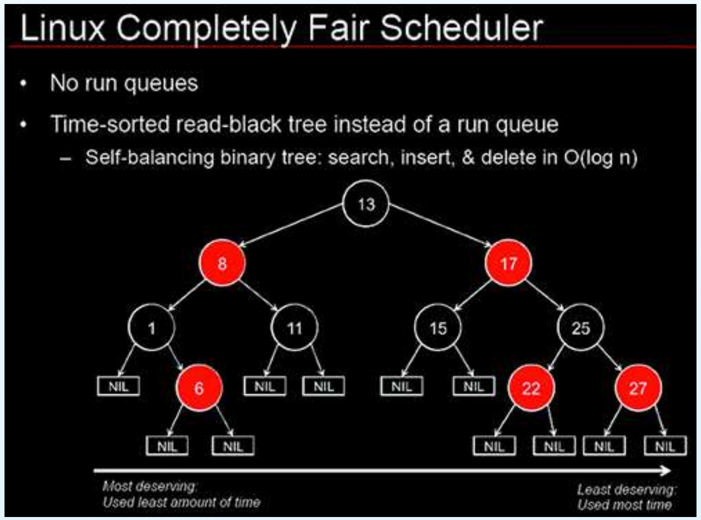
- Every single process p runs either in **Ring 0** or **Ring 3**.
  - **Ring 0**: *low-level or hardware tasks.*
  - **Ring 3**: *user-space tasks.*
- p runs only for a measurable time t.
- The **GNU/Linux Scheduler** is in charge of making p yield the cpu.
  - Modern GNU/Linux Kernels implements the **CFS** scheduler.
  - The Kernel executes `schedule()` to call the Scheduler.



Context Switching Basis

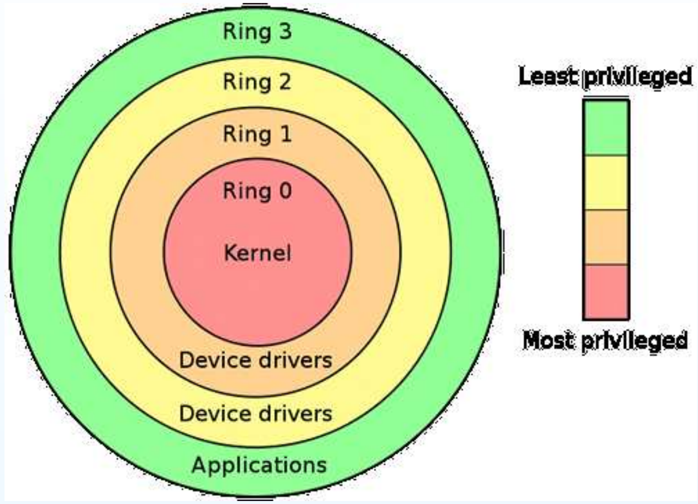
- Every single process p runs either in **Ring 0** or **Ring 3**.
  - **Ring 0**: *low-level or hardware tasks.*
  - **Ring 3**: *user-space tasks.*
- p runs only for a measurable time t.
- The **GNU/Linux Scheduler** is in charge of making p yield the cpu.
  - Modern GNU/Linux Kernels implements the **CFS** scheduler.
  - The Kernel executes `schedule()` to call the Scheduler.

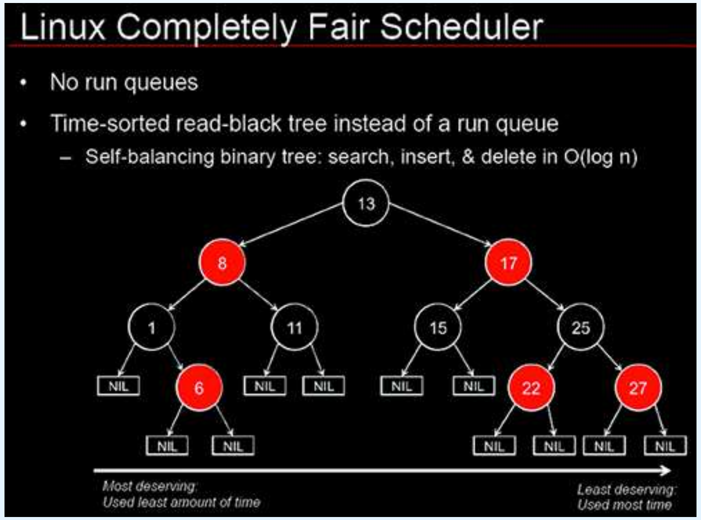




Context Switching Basis

- Every single process p runs either in **Ring 0** or **Ring 3**.
  - **Ring 0**: *low-level or hardware tasks.*
  - **Ring 3**: *user-space tasks.*
- p runs only for a measurable time t.
- The **GNU/Linux Scheduler** is in charge of making p yield the cpu.
  - Modern GNU/Linux Kernels implements the **CFS** scheduler.
  - The Kernel executes `schedule()` to call the Scheduler.





Context Switching Basis

- Whenever p yields the processor, there is a **Context Switch**.

Context Switching Basis

- Whenever p yields the processor, there is a **Context Switch**.
  - ***Voluntary***:
    - p has issued a **System Call**, and now the Kernel is returning from it.
    - p has issued an explicit call to `schedule()`, by means of calling `sched_yield()`.
    - p has ended its execution.

Context Switching Basis

- Whenever p yields the processor, there is a **Context Switch**.
  - ***Voluntary***:
    - p has issued a **System Call**, and now the Kernel is returning from it.
    - p has issued an explicit call to `schedule()`, by means of calling `sched_yield()`.
    - p has ended its execution.

Context Switching Basis

- Whenever p yields the processor, there is a **Context Switch**.
  - ***Voluntary***:
    - p has issued a **System Call**, and now the Kernel is returning from it.
    - p has issued an explicit call to `schedule()`, by means of calling `sched_yield()`.
    - p has ended its execution.

Context Switching Basis

- Whenever p yields the processor, there is a **Context Switch**.
  - ***Voluntary***:
    - p has issued a **System Call**, and now the Kernel is returning from it.
    - p has issued an explicit call to `schedule()`, by means of calling `sched_yield()`.
    - p has ended its execution.

Context Switching Basis

- Whenever p yields the processor, there is a **Context Switch**.
  - ***Voluntary***:
    - p has issued a **System Call**, and now the Kernel is returning from it.
    - p has issued an explicit call to `schedule()`, by means of calling `sched_yield()`.
    - p has ended its execution.
  - ***Involuntary***:
    - p has been preempted by an **Interrupt Handler**.
    - p has exhausted its processor's time proportion.
    - p has been preempted in favour of a recently waken up task with a higher priority.

Context Switching Basis

- Whenever p yields the processor, there is a **Context Switch**.
  - ***Voluntary***:
    - p has issued a **System Call**, and now the Kernel is returning from it.
    - p has issued an explicit call to `schedule()`, by means of calling `sched_yield()`.
    - p has ended its execution.
  - ***Involuntary***:
    - p has been preempted by an **Interrupt Handler**.
    - p has exhausted its processor's time proportion.
    - p has been preempted in favour of a recently waken up task with a higher priority.

Context Switching Basis

- Whenever p yields the processor, there is a **Context Switch**.
  - ***Voluntary***:
    - p has issued a **System Call**, and now the Kernel is returning from it.
    - p has issued an explicit call to `schedule()`, by means of calling `sched_yield()`.
    - p has ended its execution.
  - ***Involuntary***:
    - p has been preempted by an **Interrupt Handler**.
    - p has exhausted its processor's time proportion.
    - p has been preempted in favour of a recently waken up task with a higher priority.

Context Switching Basis

- Whenever p yields the processor, there is a **Context Switch**.
  - ***Voluntary***:
    - p has issued a **System Call**, and now the Kernel is returning from it.
    - p has issued an explicit call to `schedule()`, by means of calling `sched_yield()`.
    - p has ended its execution.
  - ***Involuntary***:
    - p has been preempted by an **Interrupt Handler**.
    - p has exhausted its processor's time proportion.
    - p has been preempted in favour of a recently waken up task with a higher priority.

Context Switching Basis

- Whenever p yields the processor, there is a **Context Switch**.
  - ***Voluntary***:
    - p has issued a **System Call**, and now the Kernel is returning from it.
    - p has issued an explicit call to `schedule()`, by means of calling `sched_yield()`.
    - p has ended its execution.
  - ***Involuntary***:
    - p has been preempted by an **Interrupt Handler**.
    - p has exhausted its processor's time proportion.
    - p has been preempted in favour of a recently waken up task with a higher priority.
- Modern GNU/Linux operating systems **does not have counters for all these particular** Context Switching sub-types.

Context Switching Basis

- Whenever p yields the processor, there is a **Context Switch**.
  - ***Voluntary***:
    - p has issued a **System Call**, and now the Kernel is returning from it.
    - p has issued an explicit call to `schedule()`, by means of calling `sched_yield()`.
    - p has ended its execution.
  - ***Involuntary***:
    - p has been preempted by an **Interrupt Handler**.
    - p has exhausted its processor's time proportion.
    - p has been preempted in favour of a recently waken up task with a higher priority.
- Modern GNU/Linux operating systems **does not have counters for all these particular** Context Switching sub-types.
- Our project **does add** these new counters.

Tools, APIs & ABIS

- **Tools**
  - Based on the **`/proc`** interface.
  - Gather **cumulative** statistics for total amount of **Voluntary** and **Involuntary** Context Switches.
  - `atsar`, reads `/proc/stat`
  - `pidstat`, reads `/proc/`*PID*`/stat`, works with **averages**.

Tools, APIs & ABIS

- **Tools**
  - Based on the **`/proc`** interface.
  - Gather **cumulative** statistics for total amount of **Voluntary** and **Involuntary** Context Switches.
  - `atsar`, reads `/proc/stat`
  - `pidstat`, reads `/proc/`*PID*`/stat`, works with **averages**.

Tools, APIs & ABIS

- **Tools**
  - Based on the **`/proc`** interface.
  - Gather **cumulative** statistics for total amount of **Voluntary** and **Involuntary** Context Switches.
  - `atsar`, reads `/proc/stat`
  - `pidstat`, reads `/proc/`*PID*`/stat`, works with **averages**.

Tools, APIs & ABIS

- **Tools**
  - Based on the `/proc` interface.
  - Gather **cumulative** statistics for total amount of **Voluntary** and **Involuntary** Context Switches.
  - `atsar`, reads `/proc/stat`
  - `pidstat`, reads `/proc/`*PID*`/stat`, works with **averages**.

```
Linux   catxarru   2.6.32-5-amd64  #1 SMP Mon Jan 16 16:22:28 UTC 2012   x86_64   03/20/2012

18:14:35  pswch/s runq nrproc lavg1 lavg5 avg15                    _procload_
18:14:36      100    1    323  0.29  0.33  0.32
18:14:37     1561    1    323  0.26  0.33  0.32
18:14:38     1872    1    323  0.26  0.33  0.32
18:14:39     1528    1    323  0.26  0.33  0.32
18:14:40     1190    1    323  0.26  0.33  0.32
```

Tools, APIs & ABIS

- **Tools**
  - Based on the `/proc` interface.
  - Gather **cumulative** statistics for total amount of **Voluntary** and **Involuntary** Context Switches.
  - `atsar`, reads `/proc/stat`
  - `pidstat`, reads `/proc/`PID`/stat`, works with **averages**.

```
Linux 2.6.32-5-amd64 (catxarru)        26/03/12        _x86_64_        (2 CPU)

18:00:49          PID   cswch/s nvcswch/s  Command
18:00:49         4152      0.02      0.01  vlc
```

Tools, APIs & ABIS

- **Tools**
  - Based on the **`/proc`** interface.
  - Gather **cumulative** statistics for total amount of **Voluntary** and **Involuntary** Context Switches.
  - `atsar`, reads `/proc/stat`
  - `pidstat`, reads `/proc/`*PID*`/stat`, works with **averages**.
- **Kernel ABIs**
  - **`/proc/`*PID*`/sched`** interface.
  - **`taskstats`** facility, using **Netlink** infrastructure.

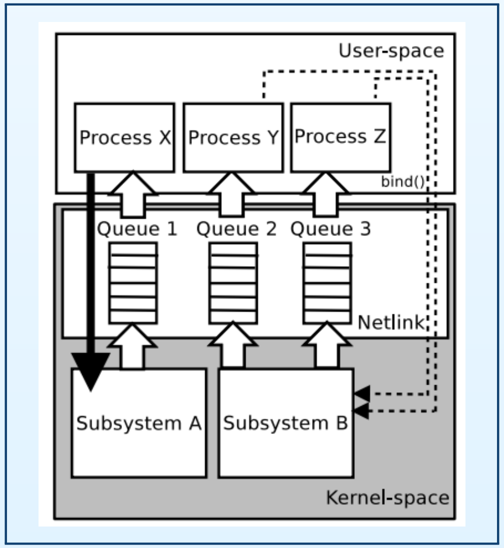
Tools, APIs & ABIS

- **Tools**
  - Based on the `/proc` interface.
  - Gather **cumulative** statistics for total amount of **Voluntary** and **Involuntary** Context Switches.
  - `atsar`, reads `/proc/stat`
  - `pidstat`, reads `/proc/`PID`/stat`, works with **averages**.
- **Kernel ABIs**
  - **`/proc/`PID`/sched`** interface.
  - **`taskstats`** facility, using **Netlink** infrastructure.

```
vlc (4152, #threads: 7)
---------------------------------------------------------
se.exec_start                     :       3196505.837931
se.vruntime                       :        370749.896090
se.sum_exec_runtime               :            72.197356
se.avg_overlap                    :             0.088572
se.avg_wakeup                     :             1.135731
se.avg_running                    :             0.030198
nr_switches                       :                  126
nr_voluntary_switches             :                  102
nr_involuntary_switches           :                   24
se.load.weight                    :                 1024
policy                            :                    0
prio                              :                  120
clock-delta                       :                  276
```

Tools, APIs & ABIS

- **Tools**
  - Based on the `/proc` interface.
  - Gather **cumulative** statistics for total amount of **Voluntary** and **Involuntary** Context Switches.
  - `atsar`, reads `/proc/stat`
  - `pidstat`, reads `/proc/`*PID*`/stat`, works with **averages**.
- **Kernel ABIs**
  - **`/proc/`*PID*`/sched`** interface.
  - **`taskstats`** facility, using **Netlink** infrastructure.



Tools, APIs & ABIS

- Advantages of using the `taskstats` interface.
  - Easy to communicate with the GNU/Linux Kernel.
  - There is no need to develop a **Linux Kernel Module**.
  - A client tool written in C running in user-space, **`getcw.c`**.

Tools, APIs & ABIS

- Advantages of using the `taskstats` interface.
  - Easy to communicate with the GNU/Linux Kernel.
  - There is no need to develop a **Linux Kernel Module**.
  - A client tool written in C running in user-space, **`getcw.c`**.

Tools, APIs & ABIS

- Advantages of using the `taskstats` interface.
  - Easy to communicate with the GNU/Linux Kernel.
  - There is no need to develop a **Linux Kernel Module**.
  - A client tool written in C running in user-space, **`getcw.c`**.

Tools, APIs & ABIS

- Advantages of using the `taskstats` interface.
  - Easy to communicate with the GNU/Linux Kernel.
  - There is no need to develop a **Linux Kernel Module**.
  - A client tool written in C running in user-space, **`getcw.c`**.

Tools, APIs & ABIS

- Advantages of using the `taskstats` interface.
  - Easy to communicate with the GNU/Linux Kernel.
  - There is no need to develop a **Linux Kernel Module**.
  - A client tool written in C running in user-space, **`getcw.c`**.
- Extending the `taskstats` interface...

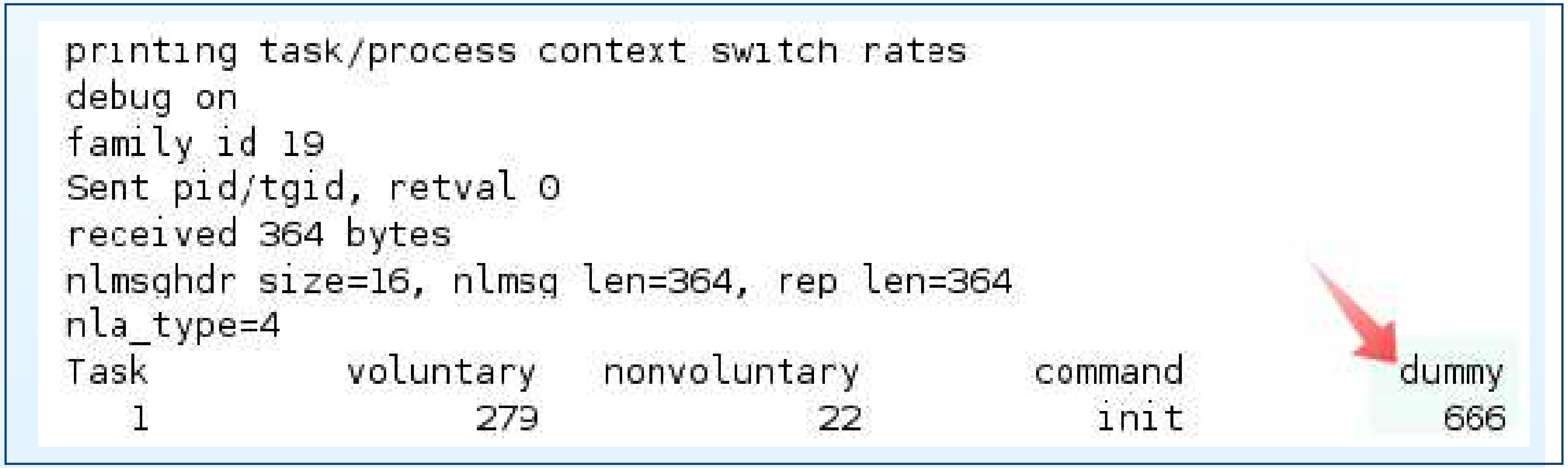
```
struct taskstats {
  ...
    /* New version: 8*/
      u64   dummy ;
}

/* Let's fill "dummy" : */
stats->dummy = 666 ;

#define MIN VERSION 8
...
/* Get dummy if we do have the right version
       ( t->version >=MIN VERSION) ? t->dummy: 0
);
...
```

Tools, APIs & ABIS

- Advantages of using the `taskstats` interface.
  - Easy to communicate with the GNU/Linux Kernel.
  - There is no need to develop a **Linux Kernel Module**.
  - A client tool written in C running in user-space, **`getcw.c`**.
- Extending the `taskstats` interface...
- ... and reading it back from user-space.



```
printing task/process context switch rates
debug on
family id 19
Sent pid/tgid, retval 0
received 364 bytes
nlmsghdr size=16, nlmsg len=364, rep len=364
nla_type=4
Task          voluntary   nonvoluntary        command         dummy
   1                279             22           init           666
```

Tools, APIs & ABIS

- Advantages of using the `taskstats` interface.
  - Easy to communicate with the GNU/Linux Kernel.
  - There is no need to develop a **Linux Kernel Module**.
  - A client tool written in C running in user-space, **`getcw.c`**.
- Extending the `taskstats` interface...
- ... and reading it back from user-space.
- The `task_struct` data structure has to be altered accordingly.

```
printing task/process context switch rates
debug on
family id 19
Sent pid/tgid, retval 0
received 364 bytes
nlmsghdr size=16, nlmsg len=364, rep len=364
nla_type=4
Task          voluntary   nonvoluntary        command        dummy
   1                279             22           init          666
```

Design & Implementation

- We have **added new counters** to the GNU/Linux Kernel.

Counter	*Index*	*Description*
`nsyscalls`	-	Total amount of issued syscalls
`nvcsw_ext[]`	0	Calls to `schedule()` at `ret_from_sys_call`.
	1	Voluntary task's exit. [1]
	2	Calls to the `sched_yield()` system call.
`nsyscalls_schedule[]`	-	Number of calls to `schedule()` per syscall.
`nivcsw_ext[]`	0	Calls to `schedule()` during `try_to_wake_up()`.
	1	Calls to `schedule()` during `do_irq()`.
`nivcsw_ext_exit`	-	Involuntary task's exit. [1]

Design & Implementation

- We have **added new counters** to the GNU/Linux Kernel.
- We have extended the `task_struct` and `taskstats` data structures to implement them.


```
struct task_struct {
...
  unsigned long nsyscalls;
  unsigned long nvcsw_ext[3];
  atomic64_t nivcsw_ext[2];
  unsigned long nivcsw_ext_exit;
  unsigned long nsyscalls_schedule[__NR_syscall_max+1];
...
}
```

Design & Implementation

- We have **added new counters** to the GNU/Linux Kernel.
- We have extended the `task_struct` and `taskstats` data structures to implement them.
- In order to read their data, we have implemented a client C program communicating with the Kernel via **NetLink**, called `getcsw.c`.

Design & Implementation

- We have **added new counters** to the GNU/Linux Kernel.
- We have extended the `task_struct` and `taskstats` data structures to implement them.
- In order to read their data, we have implemented a client C program communicating with the Kernel via **NetLink**, called `getcsw.c`.
- Our counters can ...
  - *account the total amount of syscalls per task.*
  - *account the total amount of calls to the `sched_yield()` function.*
  - *account the total feasible amount of **context switches** whilst returning from a system call.*
  - *account preemption due to **Interrupts**.*
  - *account preemption due to `try_to_wake_up()`.*
  - *determine whether p has ended its execution **on its own accord** or not.*

Design & Implementation

- We have **added new counters** to the GNU/Linux Kernel.
- We have extended the `task_struct` and `taskstats` data structures to implement them.
- In order to read their data, we have implemented a client C program communicating with the Kernel via **NetLink**, called `getcsw.c`.
- Our counters can ...
  - *account the total amount of syscalls per task.*
  - *account the total amount of calls to the `sched_yield()` function.*
  - *account the total feasible amount of **context switches** whilst returning from a system call.*
  - *account preemption due to **Interrupts**.*
  - *account preemption due to `try_to_wake_up()`.*
  - *determine whether p has ended its execution **on its own accord** or not.*

Design & Implementation

- We have **added new counters** to the GNU/Linux Kernel.
- We have extended the `task_struct` and `taskstats` data structures to implement them.
- In order to read their data, we have implemented a client C program communicating with the Kernel via **NetLink**, called `getcsw.c`.
- Our counters can ...
  - *account the total amount of syscalls per task.*
  - *account the total amount of calls to the `sched_yield()` function.*
  - *account the total feasible amount of **context switches** whilst returning from a system call.*
  - *account preemption due to **Interrupts**.*
  - *account preemption due to `try_to_wake_up()`.*
  - *determine whether p has ended its execution **on its own accord** or not.*

Design & Implementation

- We have **added new counters** to the GNU/Linux Kernel.
- We have extended the `task_struct` and `taskstats` data structures to implement them.
- In order to read their data, we have implemented a client C program communicating with the Kernel via **NetLink**, called `getcsw.c`.
- Our counters can ...
  - *account the total amount of syscalls per task.*
  - *account the total amount of calls to the `sched_yield()` function.*
  - *account the total feasible amount of **context switches** whilst returning from a system call.*
  - *account preemption due to **Interrupts**.*
  - *account preemption due to `try_to_wake_up()`.*
  - *determine whether p has ended its execution **on its own accord** or not.*

Design & Implementation

- We have **added new counters** to the GNU/Linux Kernel.
- We have extended the `task_struct` and `taskstats` data structures to implement them.
- In order to read their data, we have implemented a client C program communicating with the Kernel via **NetLink**, called `getcsw.c`.
- Our counters can ...
  - *account the total amount of syscalls per task.*
  - *account the total amount of calls to the `sched_yield()` function.*
  - *account the total feasible amount of **context switches** whilst returning from a system call.*
  - *account preemption due to **Interrupts**.*
  - *account preemption due to `try_to_wake_up()`.*
  - *determine whether p has ended its execution **on its own accord** or not.*

Design & Implementation

- We have **added new counters** to the GNU/Linux Kernel.
- We have extended the `task_struct` and `taskstats` data structures to implement them.
- In order to read their data, we have implemented a client C program communicating with the Kernel via **NetLink**, called `getcsw.c`.
- Our counters can ...
  - *account the total amount of syscalls per task.*
  - *account the total amount of calls to the `sched_yield()` function.*
  - *account the total feasible amount of **context switches** whilst returning from a system call.*
  - *account preemption due to **Interrupts**.*
  - *account preemption due to `try_to_wake_up()`.*
  - *determine whether p has ended its execution **on its own accord** or not.*

Design & Implementation

- We have **added new counters** to the GNU/Linux Kernel.
- We have extended the `task_struct` and `taskstats` data structures to implement them.
- In order to read their data, we have implemented a client C program communicating with the Kernel via **NetLink**, called `getcsw.c`.
- Our counters can ...
  - *account the total amount of syscalls per task.*
  - *account the total amount of calls to the `sched_yield()` function.*
  - *account the total feasible amount of **context switches** whilst returning from a system call.*
  - *account preemption due to **Interrupts**.*
  - *account preemption due to `try_to_wake_up()`.*
  - *determine whether p has ended its execution **on its own accord** or not.*

Design & Implementation

- We have **added new counters** to the GNU/Linux Kernel.
- We have extended the `task_struct` and `taskstats` data structures to implement them.
- In order to read their data, we have implemented a client C program communicating with the Kernel via **NetLink**, called `getcsw.c`.
- Our counters can ...
  - *account the total amount of syscalls per task.*
  - *account the total amount of calls to the `sched_yield()` function.*
  - *account the total feasible amount of **context switches** whilst returning from a system call.*
  - *account preemption due to **Interrupts**.*
  - *account preemption due to `try_to_wake_up()`.*
  - *determine whether p has ended its execution **on its own accord** or not.*
- To enable these counters, a **patch** has to be applied to the GNU/Linux Kernel.

Execution Examples

- Our main interface to the Kernel is `getcsw.c`.

Flag	*Description*	*Sample*
`-p` *PID*	Mandatory. Gets process *PID*'s stats.	`./getcsw -p 2345`
`-i`	Shows Involuntary Context Switches extended counters.	`./getcsw -p 2345 -i`
`-l`	Gets process *PID*'s stats at infinite intervals of 1s.	`./getcsw -p 1 -l`
`-d` *t*	Alters the default `-l`'s delay of 1s to *t* seconds.	`./getcsw -p 45 -l -d 10`
`-m` *mask*	Sets which cpu(s) we are listening to finishing tasks.	`./getcsw -p 1 -m "0,3"`
`-v`	Enables verbosity.	`./getcsw -p 2345 -l -v`

Execution Examples

- Our main interface to the Kernel is `getcsw.c`.

Flag	*Description*	*Sample*
`-p` *PID*	Mandatory. Gets process *PID*'s stats.	`./getcsw -p 2345`
`-i`	Shows Involuntary Context Switches extended counters.	`./getcsw -p 2345 -i`
`-l`	Gets process *PID*'s stats at infinite intervals of 1s.	`./getcsw -p 1 -l`
`-d` *t*	Alters the default `-l`'s delay of 1s to *t* seconds.	`./getcsw -p 45 -l -d 10`
`-m` *mask*	Sets which cpu(s) we are listening to finishing tasks.	`./getcsw -p 1 -m "0,3"`
`-v`	Enables verbosity.	`./getcsw -p 2345 -l -v`

Having a peak at the counters

```
./getcsw -p `ps -C scp|tail -1|cut -d" " -f2`
Task   voluntary   nonvoluntary   syscalls   ret.syscalls   vol.sched   vol.exit
2880        1035             32       9266             19        1016          -
```

Execution Examples

- Our main interface to the Kernel is `getcsw.c`.

Flag	*Description*	*Sample*
`-p` *PID*	Mandatory. Gets process *PID*'s stats.	`./getcsw -p 2345`
`-i`	Shows Involuntary Context Switches extended counters.	`./getcsw -p 2345 -i`
`-l`	Gets process *PID*'s stats at infinite intervals of 1s.	`./getcsw -p 1 -l`
`-d` *t*	Alters the default `-l`'s delay of 1s to *t* seconds.	`./getcsw -p 45 -l -d 10`
`-m` *mask*	Sets which cpu(s) we are listening to finishing tasks.	`./getcsw -p 1 -m "0,3"`
`-v`	Enables verbosity.	`./getcsw -p 2345 -l -v`

Reading the Involuntary Context Switches extended counters

```
./getcsw -p 1 -i
Task      voluntary   nonvoluntary   wake_up   IRQS
   1          71586            142       139     64
```

Execution Examples

- Our main interface to the Kernel is `getcsw.c`.

Flag	*Description*	*Sample*
`-p` *PID*	Mandatory. Gets process *PID*'s stats.	`./getcsw -p 2345`
`-i`	Shows Involuntary Context Switches extended counters.	`./getcsw -p 2345 -i`
`-l`	Gets process *PID*'s stats at infinite intervals of 1s.	`./getcsw -p 1 -l`
`-d` *t*	Alters the default `-l`'s delay of 1s to *t* seconds.	`./getcsw -p 45 -l -d 10`
`-m` *mask*	Sets which cpu(s) we are listening to finishing tasks.	`./getcsw -p 1 -m "0,3"`
`-v`	Enables verbosity.	`./getcsw -p 2345 -l -v`

Reading the counters at infinite intervals of time $t = 10$ seconds.

```
./getcsw -p `ps -C ping|tail -1|cut -d" " -f2` -l -d 10
Task   voluntary   nonvoluntary   syscalls   ret.syscalls   vol.sched   vol.exit
1417          42              3        473              2          40          -
1417         191              5       1667              4         187          -
1417         281              5       2221              4         277          -
...
```

Execution Examples

- Our main interface to the Kernel is `getcsw.c`.

Flag	*Description*	*Sample*
`-p` *PID*	Mandatory. Gets process *PID*'s stats.	`./getcsw -p 2345`
`-i`	Shows Involuntary Context Switches extended counters.	`./getcsw -p 2345 -i`
`-l`	Gets process *PID*'s stats at infinite intervals of 1s.	`./getcsw -p 1 -l`
`-d` *t*	Alters the default `-l`'s delay of 1s to *t* seconds.	`./getcsw -p 45 -l -d 10`
`-m` *mask*	Sets which cpu(s) we are listening to finishing tasks.	`./getcsw -p 1 -m "0,3"`
`-v`	Enables verbosity.	`./getcsw -p 2345 -l -v`

Waiting for a task to end

```
./getcsw -p `ps -C ping|tail -1|cut -d" " -f2` -m "0-3"
Task   voluntary   nonvoluntary   syscalls   ret.syscalls   vol.sched   vol.exit
3315          96              2        901              1          95          y
```

Execution Examples

- Our main interface to the Kernel is `getcsw.c`.

Flag	*Description*	*Sample*
`-p` *PID*	Mandatory. Gets process *PID*'s stats.	`./getcsw -p 2345`
`-i`	Shows Involuntary Context Switches extended counters.	`./getcsw -p 2345 -i`
`-l`	Gets process *PID*'s stats at infinite intervals of 1s.	`./getcsw -p 1 -l`
`-d` *t*	Alters the default `-l`'s delay of 1s to *t* seconds.	`./getcsw -p 45 -l -d 10`
`-m` *mask*	Sets which cpu(s) we are listening to finishing tasks.	`./getcsw -p 1 -m "0,3"`
`-v`	Enables verbosity.	`./getcsw -p 2345 -l -v`

- We have written some trivial **scripts** to add more functionality:

Execution Examples

- Our main interface to the Kernel is `getcsw.c`.

Flag	*Description*	*Sample*
`-p` *PID*	Mandatory. Gets process *PID*'s stats.	`./getcsw -p 2345`
`-i`	Shows Involuntary Context Switches extended counters.	`./getcsw -p 2345 -i`
`-l`	Gets process *PID*'s stats at infinite intervals of 1s.	`./getcsw -p 1 -l`
`-d` *t*	Alters the default `-l`'s delay of 1s to *t* seconds.	`./getcsw -p 45 -l -d 10`
`-m` *mask*	Sets which cpu(s) we are listening to finishing tasks.	`./getcsw -p 1 -m "0,3"`
`-v`	Enables verbosity.	`./getcsw -p 2345 -l -v`

- We have written some trivial **scripts** to add more functionality:
  - *`fschedyield.sh`, in charge of looking for running tasks that are calling the `sched_yield()` function.*

```
Task          Calls          Command
5708          1              /usr/sbin/kerneloops
6432          3              ./io
```

Execution Examples

- Our main interface to the Kernel is `getcsw.c`.

Flag	*Description*	*Sample*
-p *PID*	Mandatory. Gets process *PID*'s stats.	./getcsw -p 2345
-i	Shows Involuntary Context Switches extended counters.	./getcsw -p 2345 -i
-l	Gets process *PID*'s stats at infinite intervals of 1s.	./getcsw -p 1 -l
-d *t*	Alters the default -l's delay of 1s to *t* seconds.	./getcsw -p 45 -l -d 10
-m *mask*	Sets which cpu(s) we are listening to finishing tasks.	./getcsw -p 1 -m "0,3"
-v	Enables verbosity.	./getcsw -p 2345 -l -v

- We have written some trivial **scripts** to add more functionality:
  - *`fschedyield.sh`, in charge of looking for running tasks that are calling the `sched_yield()` function.*
  - *`fcalltable.sh`, in charge of building a table of calls to the scheduler per each system call during their return.*

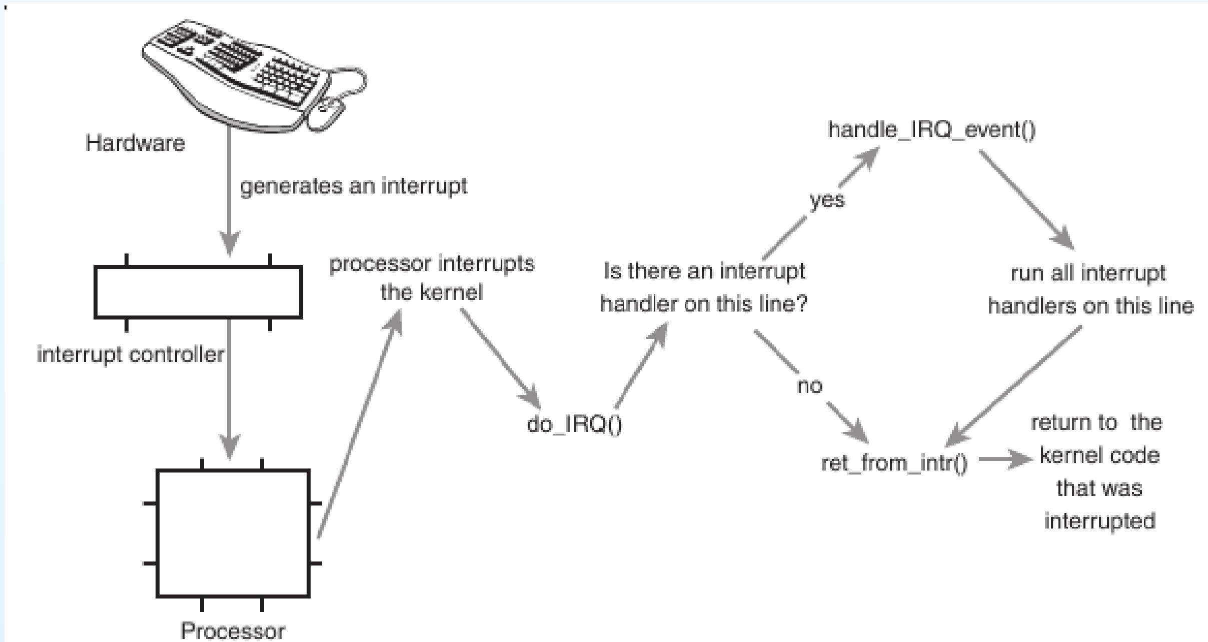
```
-------------------------------------------
Summary
-------------------------------------------
System Call                Calls to schedule()
read                                   230
write                                   29
close                                    2
mmap                                     1
mprotect                                 1
```

Generating Hardware Interrupts at will

- An **Interrupt** always preempts p.

Generating Hardware Interrupts at will

- An **Interrupt** always preempts p.
- Whenever an Interrupt is raised, the **GNU/Linux Kernel** handles it by calling `do_IRQ()`.



Generating Hardware Interrupts at will

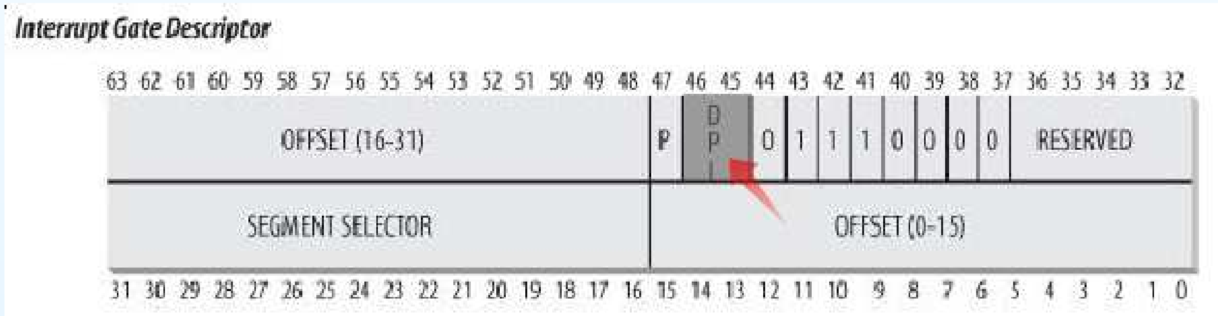
- An **Interrupt** always preempts p.
- Whenever an Interrupt is raised, the **GNU/Linux Kernel** handles it by calling `do_IRQ()`.
- Most Interrupts are **maskable**: they can be ignored.

Generating Hardware Interrupts at will

- An **Interrupt** always preempts p.
- Whenever an Interrupt is raised, the **GNU/Linux Kernel** handles it by calling `do_IRQ()`.
- Most Interrupts are **maskable**: they can be ignored.
- **NMIs** cannot be ignored; they are ideal to test our project.

Generating Hardware Interrupts at will

- An **Interrupt** always preempts p.
- Whenever an Interrupt is raised, the **GNU/Linux Kernel** handles it by calling `do_IRQ()`.
- Most Interrupts are **maskable**: they can be ignored.
- **NMIs** cannot be ignored; they are ideal to test our project.
- To generate NMIs at will, we need to alter the **Kernel IDT**, so that `int $0x2` can be executed with $DPL = 3$.



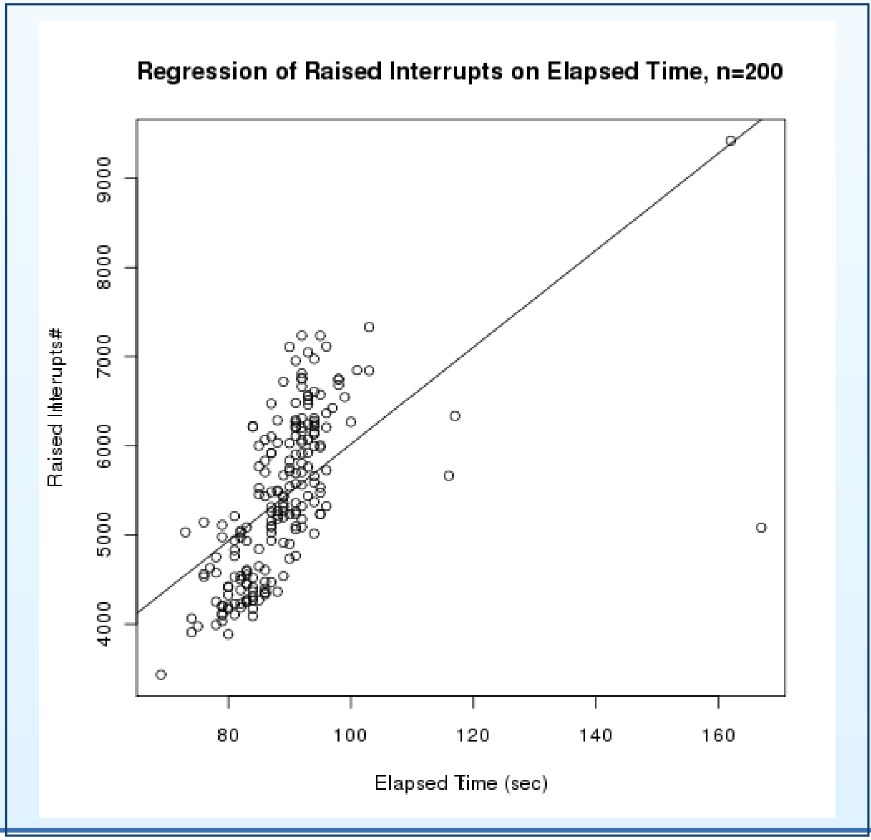

```
static inline void _set_gate(int gate, unsigned type, void *addr,
                             unsigned dpl, unsigned ist, unsigned seg)
{
    gate_desc s;
    pack_gate(&s, type, (unsigned long)addr, (gate==2)?3:dpl, ist, seg);
    write_idt_entry(idt_table, gate, &s);
}
```

Preliminary Results

- By accounting preemption due to `do_IRQ()`, we can determine whenever there is **an external problem** affecting p's throughput.

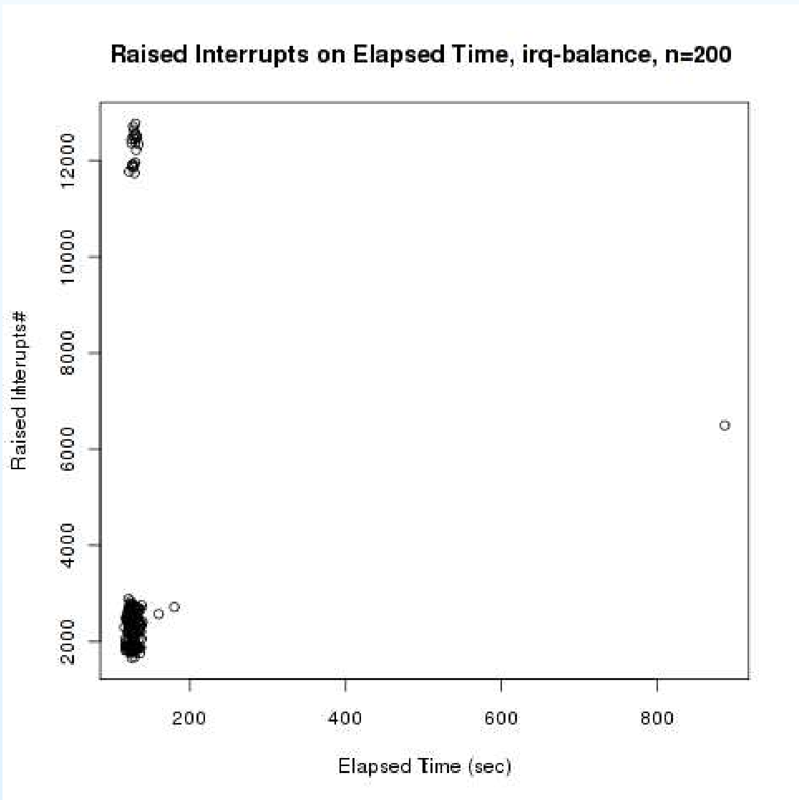
Preliminary Results

- By accounting preemption due to `do_IRQ()`, we can determine whenever there is **an external problem** affecting p's throughput.
- As $IRQS \longrightarrow \infty$, the time spent by p increases.



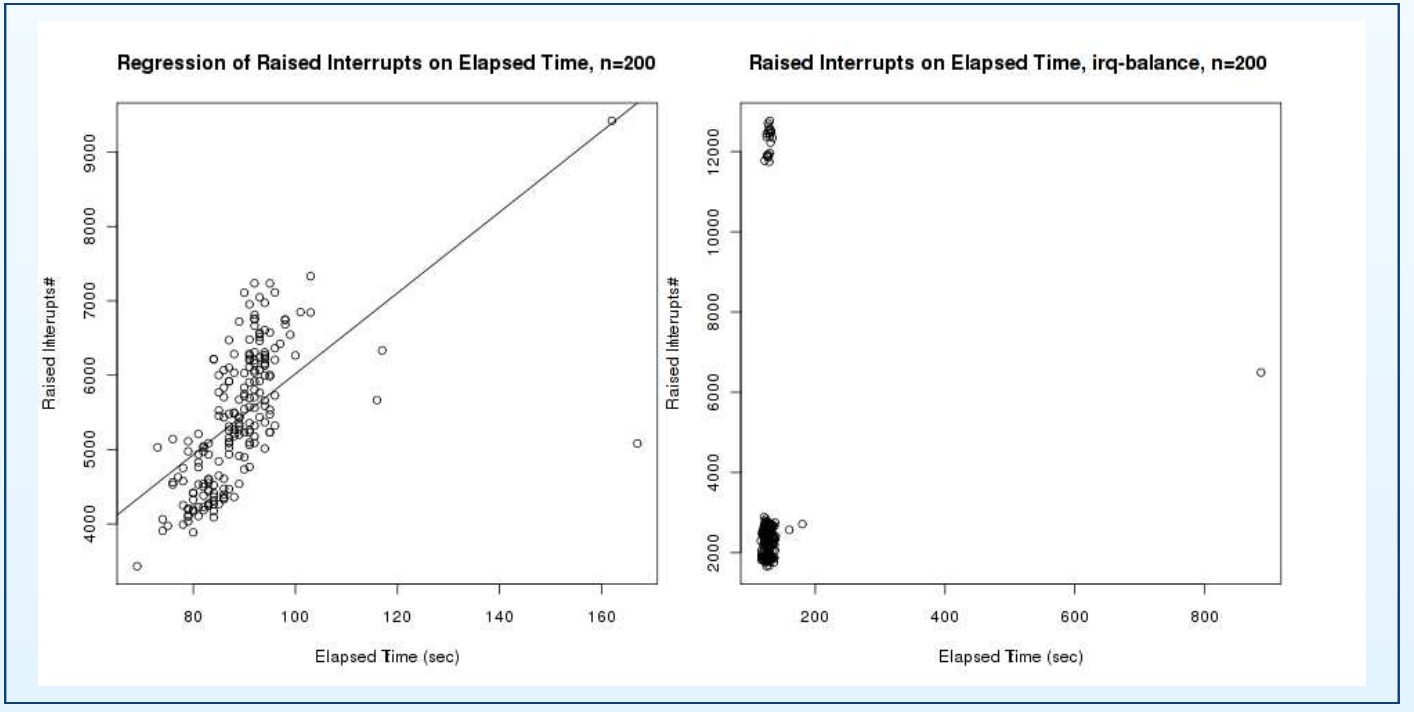
Preliminary Results

- By accounting preemption due to `do_IRQ()`, we can determine whenever there is **an external problem** affecting p's throughput.
- As $IRQS \longrightarrow \infty$, the time spent by p increases.
- `IRQ-balance` can help to diminish this time by **reducing the number of interrupts** preempting p.



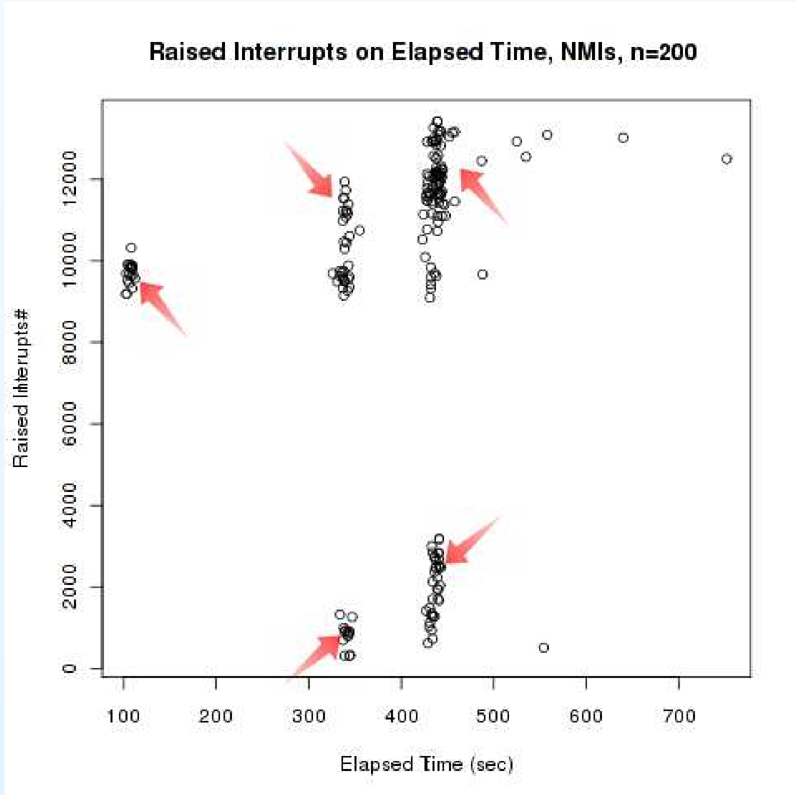
Preliminary Results

- By accounting preemption due to `do_IRQ()`, we can determine whenever there is **an external problem** affecting p's throughput.
- As $IRQS \longrightarrow \infty$, the time spent by p increases.
- `IRQ-balance` can help to diminish this time by **reducing the number of interrupts** preempting p.



Preliminary Results

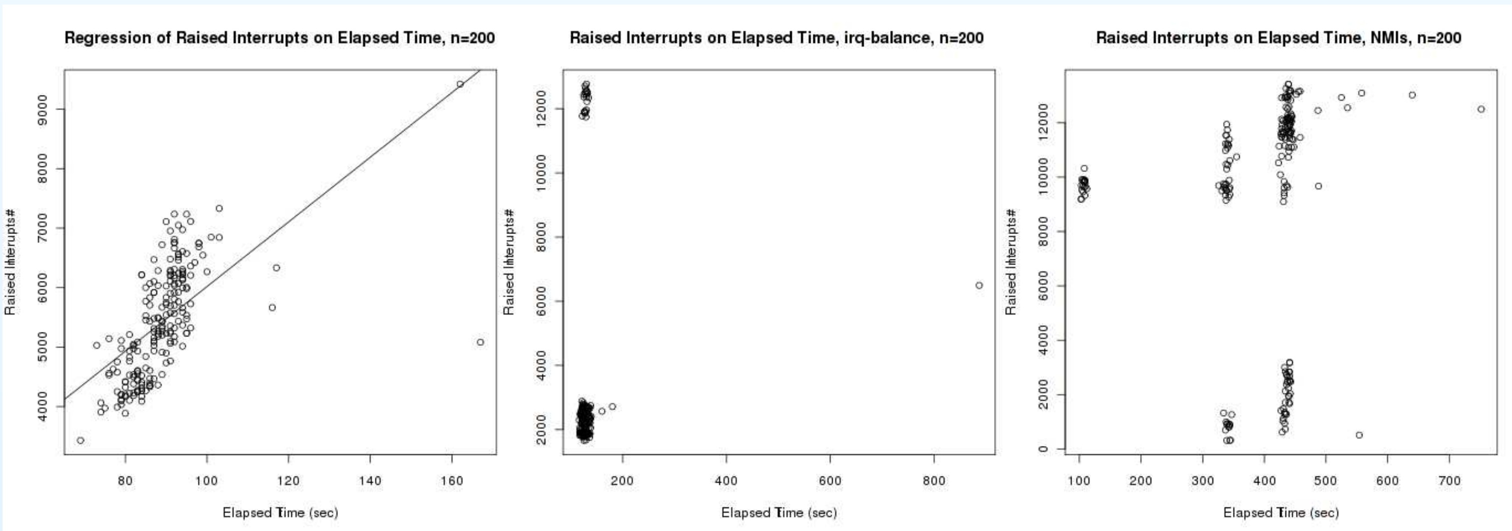
- Whenever there is a **hardware malfunction, the data starts being chaotic**.



Preliminary Results

- Whenever there is a **hardware malfunction**, **the data starts being chaotic**.
- It's proved that, whenever p **yields the processor due to a raised interrupt**, at some measurable intervals of time t_i, its time spent in doing its job increases.

	Normal	irq-balance	NMIs
$t(s)$	17779	26214	75132
$IRQs\#$	1082867	659144	1701326



Future Work

- Architecture-independent,
- Re-ran the experiments adding some entropy for larger values of n,
- Extend `getcsw` to integrate it with plot facilities,
- Analyse the real impact of our counters inside the Scheduler,
- ...

Future Work

- Architecture-independent,
- Re-ran the experiments adding some entropy for larger values of n,
- Extend `getcsw` to integrate it with plot facilities,
- Analyse the real impact of our counters inside the Scheduler,
- ...

Future Work

- Architecture-independent,
- Re-ran the experiments adding some entropy for larger values of n,
- Extend `getcsw` to integrate it with plot facilities,
- Analyse the real impact of our counters inside the Scheduler,
- ...

Future Work

- Architecture-independent,
- Re-ran the experiments adding some entropy for larger values of n,
- Extend `getcsw` to integrate it with plot facilities,
- Analyse the real impact of our counters inside the Scheduler,
- ...

Future Work

- Architecture-independent,
- Re-ran the experiments adding some entropy for larger values of n,
- Extend `getcsw` to integrate it with plot facilities,
- Analyse the real impact of our counters inside the Scheduler,
- ...

Thanks for coming!

Universitat Oberta de Catalunya

www.uoc.edu

`toni.castillo@uoc.edu`