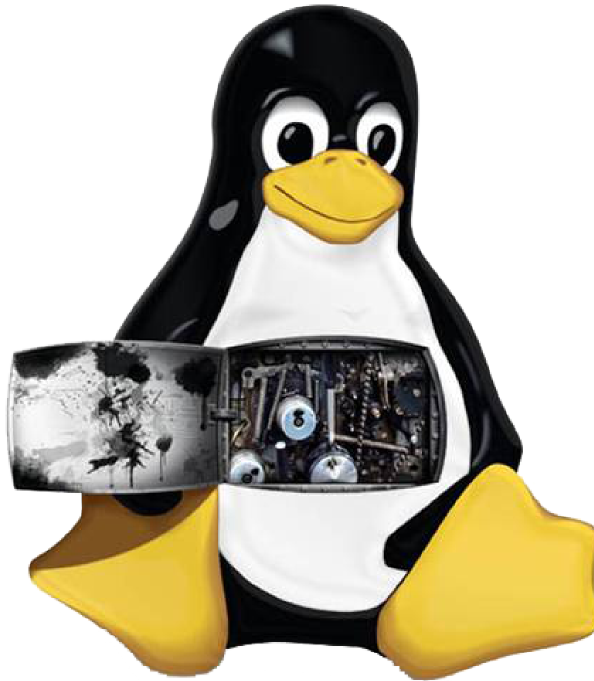
Linux Kernel Development (LKD)

Session 2



CISTER Framework: Laboratory 2

Paulo Baltarejo Sousa

pbs@isep.ipp.pt

2017

CISTER - Research Centre in Real-Time & Embedded Computing Systems

isep Instituto Superior de Engenharia do Porto

INESCTEC

1 Introduction

The goal of the CISTER framework is to create a set of tools that help CISTER Summer School Internship students to explore and to learn the how to develop code for Linux kernel.

2 Preparing

One of the goals is to avoid spreading out files for many directories in the Linux kernel source code tree. So, it will be created a directory, called `cister` in the Linux kernel to encompass all files required for such a framework. For that purpose, open a terminal and type:

```
> cd kernel_sc
> cd linux-4.12.4-cister
> cd kernel
> mkdir cister
```

3 Configuration menu

The next step is to add a new configuration option entry that is used to wrap all the CISTER framework code. Then, the `Kconfig` file has to be created in the `linux-4.12.4-cister/kernel/cister` directory. The content of the `Kconfig` file is:

```
menu "CISTER framework"
config CISTER_FRAMEWORK
  bool "CISTER Framework"
  default y
endmenu
```

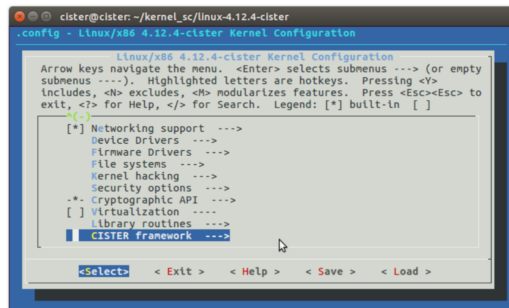
In spite of the entry name is `CISTER_FRAMEWORK`, in the code it will be referred as `CONFIG_CISTER_FRAMEWORK`. The `CONFIG_` prefix is assumed but is not written. The `bool` directive states that this option entry is a feature and can only be set using two values (y or n). The quoted text following the directive provides the name of this option in the various configuration utilities, like `make menuconfig`. The default value of this option is defined using `default` directive.

Since the host system used in this document is based on a x86 architecture, the `linux-4.12.4-cister/kernel/cister/Kconfig` configuration file must be included into the root configuration file for x86 architecture located

at `linux-4.12.4-cister/arch/x86/Kconfig` file.

```
...
source "crypto/Kconfig"

source "arch/x86/kvm/Kconfig"

source "lib/Kconfig"

source "kernel/cister/Kconfig"
```

Open a terminal, move to the `linux-4.12.4-cister` directory and type:
`> make menuconfig`
Use the "Down" arrow key to move to the last option, called "CISTER Framework" that you have just created.



4 Preparing

As mentioned before, all CISTER framework implementation source code files will be placed into the `linux-4.12.4-cister/kernel/cister` directory.

The next step is to create a `Makefile` file for compiling the CISTER Framework source code. Create an empty file called `Makefile` in the `linux-4.12.4-cister/kernel/cister` directory.

Next, change the `linux-4.12.4-cister/kernel/Makefile` file to include `cister` directory in the compilation process:

```
...
obj-$(CONFIG_MEMBARRIER) += membarrier.o

obj-$(CONFIG_HAS_IOMEM) += memremap.o

$(obj)/configs.o: $(obj)/config_data.h

targets += config_data.gz
$(obj)/config_data.gz: $(KCONFIG_CONFIG) FORCE
  $(call if_changed,gzip)

      filechk_ikconfiggz = (echo "static const char kernel_config_data[] __used = MAGIC_START"; cat $< |
             scripts/basic/bin2c; echo "MAGIC_END;")
targets += config_data.h
$(obj)/config_data.h: $(obj)/config_data.gz FORCE
  $(call filechk,ikconfiggz)

#including CISTER Framework in the compilation process
obj-$(CONFIG_CISTER_FRAMEWORK) += cister/
```

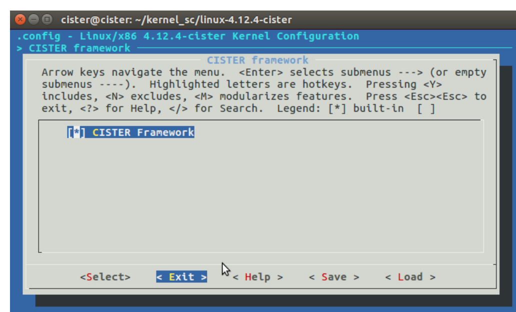
CORRIGIR Now, you are ready to compile the Linux kernel. Open a terminal and move to the Linux kernel source code parent directory:

```
> cd kernel_sc
> sudo ./kcompile.sh
```

```
> cd kernel_sc
> cd linux-4.12.4-cister
```

```
> make menuconfig
```

Check if the CISTER Framework is included.



5 Scheduling Tracing Mechanism

The goal of this feature is to create scheduling tracing mechanism in the Linux kernel. The purpose is to register the event types: SCHED_TICK, SWITCH_TO and SWITCH_AWAY.

The SCHED_TICK event happens whenever the periodic interrupt, called tick, is fired. Note that, there is always a task executing when such event occurs. The SWITCH_TO event happens whenever a task is assigned to CPU for execution, while SWITCH_AWAY event happens whenever a task is removed from CPU.

It must be created the CISTER_TRACING configuration entry that depends on CISTER_FRAMEWORK entry.

This feature must be a module, because it will be created an entry in the proc directory to pass the scheduling information from kernel to user space. So, the interaction will be through a proc entry. So, it must be created a proc/cister_trace entry. To store the scheduling data, this module must implement a ring buffer. It must save the the following information: event type, event timestamp, task policy, task prio, task state, task pid and task comm.

1. Open the linux-4.12.4-cister/kernel/cister/Kconfig file and update it by adding:

```
menu "CISTER framework"
config CISTER_FRAMEWORK
  bool "CISTER Framework"
  default y

config CISTER_TRACING
  bool "CISTER tracing"
  default y
  depends on CISTER_FRAMEWORK

endmenu
```

2. Create the trace.h file in the linux-4.12.4-cister/kernel/cister/ directory and code with:

```
#ifndef __TRACE_H_
#define __TRACE_H_

#include <linux/sched.h>


#define TRACE_ENTRY_NAME  "cister_trace"
#define TRACE_BUFFER_SIZE  1000
#define TRACE_BUFFER_LEN 200

enum evt{
  SCHED_TICK = 0,
  SWITCH_AWAY,
  SWITCH_TO,
};

#define TRACE_TASK_COMM_LEN  16

struct trace_evt{
  enum evt event;
  unsigned long long time;
  pid_t pid;
  int state;
  int prio;
  int policy;
  char comm[TRACE_TASK_COMM_LEN];
};

struct trace_evt_buffer{
  struct trace_evt events[TRACE_BUFFER_SIZE];
  int write_item;
  int read_item;
  spinlock_t lock;
};
void cister_trace(enum evt event, struct task_struct *p);

#endif
```

3. Create the `trace.c` file in the `linux-4.12.4-cister/kernel/ciste r/` directory and code with:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/uaccess.h>

#include "trace.h"

///////////////////////////////////////////////////////////////////////
// Ring Buffer implementation
struct trace_evt_buffer trace;

unsigned char enabled = 0;

static void increment(int * item)
{
  *item = *item + 1;
  if(*item >= TRACE_BUFFER_SIZE){
    *item = 0;
  }
}
static int is_empty(int r, int w)
{
  return !(r ^ w); //xor
```

```
}
static int is_full(int r, int w)
{
  int write = w;
  increment(&write);
  return write == r;
}
static int dequeue (char *buffer)
{
  int ret = 0, len;
  char evt[20];
  spin_lock(&trace.lock);
  if(!is_empty(trace.read_item,trace.write_item)){ //if it is not empty

    switch((int)trace.events[trace.read_item].event){
      case SCHED_TICK:
        strcpy(evt,"SCH_TK");
      break;
      case SWITCH_AWAY:
        strcpy(evt,"SWT_AY");
      break;
      case SWITCH_TO:
        strcpy(evt,"SWT_TO");
      break;

    }

    len = sprintf(buffer,"%llu,",trace.events[trace.read_item].time);
    len += sprintf(buffer+len,"%s,",evt);
    len += sprintf(buffer+len,"pid,%d,",(int)trace.events[trace.read_item].pid);
    len += sprintf(buffer+len,"prio,%d,",(int)trace.events[trace.read_item].prio);
    len += sprintf(buffer+len,"policy,%d,",(int)trace.events[trace.read_item].policy);
    len += sprintf(buffer+len,"state,%d,",(int)trace.events[trace.read_item].state);
    len += sprintf(buffer+len,"%s\n",trace.events[trace.read_item].comm);


    increment(&trace.read_item);
    ret = 1;
  }
  spin_unlock(&trace.lock);
  return ret;
}

static int enqueue (enum evt event, unsigned long long time, struct task_struct *p)
{
  spin_lock(&trace.lock);
  if(is_full(trace.read_item, trace.write_item))
    increment(&trace.read_item);

  trace.events[trace.write_item].event  = event;
  trace.events[trace.write_item].time  = time;
  trace.events[trace.write_item].pid  = p->pid;
  trace.events[trace.write_item].state  = p->state;
  trace.events[trace.write_item].prio  = p->prio;
  trace.events[trace.write_item].policy  = p->policy;
  strcpy(trace.events[trace.write_item].comm, p->comm);

  increment(&trace.write_item);
  spin_unlock(&trace.lock);
  return 1;
}
ssize_t trace_read(struct file *filp, char __user *buf, size_t count, loff_t *f_pos)
{
  char buffer[TRACE_BUFFER_LEN];
  int ret = 0, len = 0;
  printk(KERN_INFO "%s:[%d] read\n",TRACE_ENTRY_NAME, current->pid);

  if(!dequeue(buffer))
    return 0;
  len = strlen(buffer);
  if(len <= 0)
    return -EFAULT;
  if(count < len)
```

```
    return -EFAULT;
  ret=copy_to_user(buf,buffer,len);
  if(ret != 0)
    return -EFAULT;
  return len;
}

static const struct file_operations trace_fops = {
  .owner    = THIS_MODULE,
  .read    = trace_read,
};

static int __init proc_trace_init(void)
{
    proc_create(TRACE_ENTRY_NAME,0444, NULL, &trace_fops);
    printk("CISTER:/proc/%s created\n", TRACE_ENTRY_NAME);
  spin_lock_init(&trace.lock);
  trace.write_item  = 0;
  trace.read_item  = 0;
  enabled   = 1;
  return 0;
}
module_init(proc_trace_init);

//This function will be used to get the event.

void cister_trace(enum evt event, struct task_struct *p)
{
  if(enabled){
    unsigned long long time = ktime_to_ns(ktime_get());
    enqueue(event, time, p);
  }
}
```

4. Open the `linux-4.12.4-cister/kernel/sched/sched.h` file and update it by adding:

```
...
#ifdef CONFIG_PARAVIRT
#include <asm/paravirt.h>
#endif

#include "cpupri.h"
#include "cpudeadline.h"
#include "cpuacct.h"

#ifdef CONFIG_SCHED_DEBUG
#define SCHED_WARN_ON(x) WARN_ONCE(x, #x)
#else
#define SCHED_WARN_ON(x) ((void)(x))
#endif

#ifdef CONFIG_CISTER_TRACING
  #include "../cister/trace.h"
#endif

struct rq;
struct cpuidle_state;
...
```

5. Open the `linux-4.12.4-cister/kernel/cister/Makefile` file and update it by adding:

```
# CISTER Framework makefile

obj-$(CONFIG_CISTER_TRACING) += trace.o
```

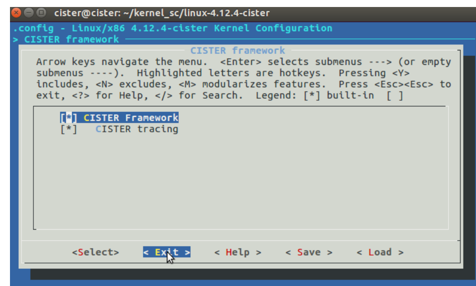
6. Now, let collect events. So, open the `linux-4.12.4-cister/kernel/sched/core.c` file and update it by adding to the `schedule` function:

```
static void __sched notrace __schedule(bool preempt)
{
  ...
  next = pick_next_task(rq, prev, &rf);
  clear_tsk_need_resched(prev);
  clear_preempt_need_resched();

  if (likely(prev != next)) {
    rq->nr_switches++;
    rq->curr = next;
    ++*switch_count;

    trace_sched_switch(preempt, prev, next);

#ifdef CONFIG_CISTER_TRACING
    cister_trace(SWITCH_AWAY, prev);
    cister_trace(SWITCH_TO, next);
#endif

    /* Also unlocks the rq: */
    rq = context_switch(rq, prev, next, &rf);
  } else {
    rq->clock_update_flags &= ~(RQCF_ACT_SKIP|RQCF_REQ_SKIP);
    rq_unlock_irq(rq, &rf);
  }

  balance_callback(rq);
}
```

7. Update the `schedule_tick` function located at `linux-4.12.4-cister/kernel/sched/core.c` file:

```
/*
 * This function gets called by the timer code, with HZ frequency.
 * We call it with interrupts disabled.
 */
void scheduler_tick(void)
{
  int cpu = smp_processor_id();
  struct rq *rq = cpu_rq(cpu);
  struct task_struct *curr = rq->curr;
  struct rq_flags rf;

  sched_clock_tick();

  rq_lock(rq, &rf);

  update_rq_clock(rq);
  curr->sched_class->task_tick(rq, curr, 0);

#ifdef CONFIG_CISTER_TRACING
  cister_trace(SCHED_TICK, curr);
#endif

  cpu_load_update_active(rq);
  calc_global_load_tick(rq);

  rq_unlock(rq, &rf);

  perf_event_task_tick();

#ifdef CONFIG_SMP
  rq->idle_balance = idle_cpu(cpu);
  trigger_load_balance(rq);
#endif
  rq_last_tick_reset(rq);
}
```

8. Check if the CISTER Tracing is included.
   `> cd kernel_sc`
   `> cd linux-4.12.4-cister`
   `> make menuconfig`



9. Compile the Linux kernel:
   `> sudo ./kcompile.sh`
   In the end of the compilation process, it is created a text file called

errors_4.12.4-cister that outputs the result of the compilation. item Reboot the system
> sudo reboot

10. Open a terminal for getting the scheduling trace:
> cat /proc/cister_trace > trace.csv
The first column refers to the timestamp and second one to the event type. The following columns presents task's data: the process identifier (pid), priority level (prio), scheduling policy (policy), process state and process name. For that it is used a tuple {description, value}.

```
59973586110,SWT_TO,pid,1325,prio,120,policy,0,state,0,compiz
59985440808,SCH_TK,pid,1325,prio,120,policy,0,state,0,compiz
59985557088,SWT_AY,pid,1325,prio,120,policy,0,state,0,compiz
59985564316,SWT_TO,pid,1043,prio,120,policy,0,state,0,notify-osd
59985627929,SWT_AY,pid,1043,prio,120,policy,0,state,1,notify-osd
59985627967,SWT_TO,pid,657,prio,120,policy,0,state,0,Xorg
59985716747,SWT_AY,pid,657,prio,120,policy,0,state,1,Xorg
59985716829,SWT_TO,pid,1043,prio,120,policy,0,state,0,notify-osd
59985731767,SWT_AY,pid,1043,prio,120,policy,0,state,1,notify-osd
59985731791,SWT_TO,pid,657,prio,120,policy,0,state,0,Xorg
59985766302,SWT_AY,pid,657,prio,120,policy,0,state,1,Xorg
59985766370,SWT_TO,pid,1043,prio,120,policy,0,state,0,notify-osd
59985787054,SWT_AY,pid,1043,prio,120,policy,0,state,1,notify-osd
59985787093,SWT_TO,pid,1325,prio,120,policy,0,state,0,compiz
59986244493,SCH_TK,pid,1325,prio,120,policy,0,state,0,compiz
59987338003,SCH_TK,pid,1325,prio,120,policy,0,state,0,compiz
59988032604,SCH_TK,pid,1325,prio,120,policy,0,state,0,compiz
59992995163,SWT_AY,pid,1325,prio,120,policy,0,state,0,compiz
59992995376,SWT_TO,pid,657,prio,120,policy,0,state,0,Xorg
59993044422,SCH_TK,pid,657,prio,120,policy,0,state,0,Xorg
59993400639,SWT_AY,pid,657,prio,120,policy,0,state,1,Xorg
59993403222,SWT_TO,pid,1325,prio,120,policy,0,state,0,compiz
59993427231,SWT_AY,pid,1325,prio,120,policy,0,state,0,compiz
59993428212,SWT_TO,pid,657,prio,120,policy,0,state,0,Xorg
59993462589,SWT_AY,pid,657,prio,120,policy,0,state,1,Xorg
59993462667,SWT_TO,pid,1325,prio,120,policy,0,state,0,compiz
59993472288,SWT_AY,pid,1325,prio,120,policy,0,state,0,compiz
59993472313,SWT_TO,pid,657,prio,120,policy,0,state,0,Xorg
```

The pid value varies from 0 to approximately 4 millions (2^{22}) and prio from 0 to 139. The policy range of values are:

```
/*
 * Scheduling policies
 */
#define SCHED_NORMAL  0
#define SCHED_FIFO  1
#define SCHED_RR  2
#define SCHED_BATCH  3
/* SCHED_ISO: reserved but not implemented yet */
#define SCHED_IDLE  5
#define SCHED_DEADLINE  6
```

The state range of values are:

```
/* Used in tsk->state: */
#define TASK_RUNNING   0
#define TASK_INTERRUPTIBLE  1
#define TASK_UNINTERRUPTIBLE  2
#define __TASK_STOPPED   4
#define __TASK_TRACED   8
/* Used in tsk->exit_state: */
#define EXIT_DEAD   16
#define EXIT_ZOMBIE   32
#define EXIT_TRACE   (EXIT_ZOMBIE | EXIT_DEAD)
/* Used in tsk->state again: */
#define TASK_DEAD   64
#define TASK_WAKEKILL   128
#define TASK_WAKING   256
#define TASK_PARKED   512
#define TASK_NOLOAD   1024
#define TASK_NEW   2048
#define TASK_STATE_MAX   4096
```