Extending the RTAI API (RTAI3.2) for LXRT:

Extending the RTAI API is easy. In my case I wanted a function returning the current unixtime in a LXRT program. Instead of adding the function to RTAI directly, I had a look at the "showroom" example "signal" and derived the following files from it. (developed on RTAI3.2-test1 and kernel 2.6.8.1 with gcc 3.3.4 on debian sarge testing)

Testing the API extension is done with the Parallel Port Interrupt LXRT Example for RTAI 3.2 - MAGMA example.

```
# make
[...]
# ./loadmods.sh
# ./parlxrtmagma
GIVE THE NUMBER OF INTERRUPTS YOU WANT TO COUNT: 2
unixtime = 1106392139025058000
RETVAL 0, OVERRUNS 0, INTERRUPT COUNT 1
unixtime = 1106392149040604000
RETVAL 0, OVERRUNS 0, INTERRUPT COUNT 2
TEST ENDS
```

The unixtime is provided in nanoseconds, so 1106392139025058000 ns = 1106392139.025058000 seconds.

Using the unixtime-extension is as easy as this:

```
#include "rtai_unixtime.h"
```

- RTAI - Real Time Linux Kernel installation (kernel 2.6.7)
- RTAI - FIFO & shared memory examples

REALTIME

- REAL TIME LINUX
- Stress tests


```
[...]
RTIME unixtime;
unixtime = rt_getunixtime();
printf("unixtime = %lld\n", unixtime);
```

Remember: RTIME is a **long long**, so we need to use **%lld** for printf'ing.

The RTAI kernel module "rt_getunixtime.c" with the additional function "static RTIME rt_getunixtime(void)":

```
/*
       RTAI kernel module rt_getunixtime.c:
       extending the API with rt_getunixtime, which will provide
       the current unixtime in the RTIME format

       Derived from the showroom example "signal" for RTAI3.2
       http://www.captain.at/programming/
 */

#include <linux/kernel.h>
#include <linux/module.h>

#include <rtai_schedcore.h>
#include "rtai_unixtime.h"

MODULE_LICENSE("GPL");

#define MODULE_NAME "RTAI_GETUNIXTIME"

static RTIME rt_getunixtime(void)
{
       struct timeval tv;
       do_gettimeofday(&tv);
       return (RTIME)( tv.tv_sec * (long long) 1000000000 + tv.tv_usec * (long long) 1000 );
}

static struct rt_fun_entry rtai_getunixtime_fun[] = {
       [GETUNIXTIME] = { 1, rt_getunixtime }
};

int init_module(void)
{
       if (set_rt_fun_ext_index(rtai_getunixtime_fun, RTAI_GETUNIXTIME_IDX)) {
              printk("Wrong index module for lxrt: %d.\n", RTAI_GETUNIXTIME_IDX);
              return -EACCES;
       }
       printk("%s: loaded.\n", MODULE_NAME);
       return 0;
}

void cleanup_module(void)
{
```

```
	reset_rt_fun_ext_index(rtai_getunixtime_fun, RTAI_GETUNIXTIME_IDX);
	printk("%s: unloaded.\n", MODULE_NAME);
}
```

The include file "rtai_unixtime.h":

```
/*
	RTAI kernel module include file rtai_unixtime.h:
	extending the API with rt_getunixtime, which will provide
	the current unixtime in the RTIME format

	Derived from the showroom example "signal" for RTAI3.2
	http://www.captain.at/programming/
 */

#define RTAI_GETUNIXTIME_IDX  1

#define GETUNIXTIME  0

#ifdef __KERNEL__

#else /* !__KERNEL__ */

#include <rtai_lxrt.h>

static inline RTIME rt_getunixtime()
{
	struct { unsigned int dummy; } arg;
	return rtai_lxrt(RTAI_GETUNIXTIME_IDX, SIZARG, GETUNIXTIME, &arg).rt;
}

#endif /* __KERNEL__ */
```

The "Makefile": (tested only with kernel 2.6 !)

```
prefix := $(shell rtai-config --prefix)

ifeq ($(prefix),)
$(error Please add <rtai-install>/bin to your PATH variable)
endif

CC = $(shell rtai-config --cc)

LXRT_CFLAGS = $(shell rtai-config --lxrt-cflags)
LXRT_LDFLAGS = $(shell rtai-config --lxrt-ldflags)

all:: parlxrtmagma

ifneq ($(findstring 2.6.,$(shell rtai-config --linux-version 2>/dev/null)),)

obj-m  := rt_getunixtime.o

KDIR   := /lib/modules/$(shell uname -r)/build
```

```
PWD             := $(shell pwd)
EXTRA_CFLAGS := -I/usr/realtime/include -I/usr/include/

all::
	$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

else

TARGET := rt_getunixtime
INCLUDE:= -I/lib/modules/`uname -r`/build/include -I/usr/realtime/include
CFLAGS := -O2 -Wall -DMODULE -DUSEFIFO -D__KERNEL__ -DLINUX
CC     := gcc

all:: rt_getunixtime.o

${TARGET}.o: ${TARGET}.c
	$(CC) $(CFLAGS) ${INCLUDE} -c ${TARGET}.c

endif

parlxrtmagma: parlxrtmagma.c
	$(CC) $(LXRT_CFLAGS) -o $@ $< $(LXRT_LDFLAGS)

clean::
	$(RM) parlxrtmagma

.PHONY: clean
```


The modified example "parlxrtmagma.c" (RTAI LXRT hard real time): see Parallel Port Interrupt LXRT Example for RTAI 3.2 - MAGMA

```
/* RTAI LXRT Parallel Port Interrupt - www.captain.at
       This examples is for RTAI3.2 (MAGMA)
       based on usi_process.c from showroom
       Needs a "null-modem" for the parallel port:
               connect any output pin (pin 2-9) with the IRQ pin (pin 10 = ACK).
*/

#include <stdio.h>
#include <errno.h>
#include <sys/mman.h>
```

```
#include <stdlib.h>
#include <unistd.h>

#include <rtai_lxrt.h>
#include <rtai_sem.h>
#include <rtai_usi.h>
#include <sys/io.h>

#include "rtai_unixtime.h"

#define PARPORT_IRQ  7
#define BASEPORT 0x378

static SEM *dspsem;
static volatile int end = 1;
static volatile int ovr, intcnt, retval, maxcnt;
static RTIME unixtime;

static void *timer_handler(void *args)
{
       RT_TASK *handler;
       if (!(handler = rt_task_init_schmod(nam2num("HANDLR"), 0, 0, 0, SCHED_FIFO, 0xF))) {
              printf("CANNOT INIT HANDLER TASK > HANDLR <\n");
              exit(1);
       }
       rt_allow_nonroot_hrt();
       mlockall(MCL_CURRENT | MCL_FUTURE);
       rt_make_hard_real_time();
       end = 0;

       rt_request_irq_task(PARPORT_IRQ, handler, RT_IRQ_TASK, 1);
       rt_startup_irq(PARPORT_IRQ);
       rt_enable_irq(PARPORT_IRQ);

       while ( !end && (ovr != RT_IRQ_TASK_ERR) ) {
              do {
                     ovr = rt_irq_wait(PARPORT_IRQ);
                     unixtime = rt_getunixtime(); // our new function
                     if (ovr == RT_IRQ_TASK_ERR) break;
                     if (end) break;
                     if (ovr > 0) {
                            // overrun processing, if any, goes here
                            rt_sem_signal(dspsem);
                     }
                     /* normal processing goes here */
                     intcnt++;
                     rt_sem_signal(dspsem);          // notify main()
                     rt_ack_irq(PARPORT_IRQ);
              } while (ovr > 0);
              rt_pend_linux_irq(PARPORT_IRQ);
       }
       rt_release_irq_task(PARPORT_IRQ);
```

```
        rt_make_soft_real_time();
        rt_task_delete(handler);
        intcnt = maxcnt;
        return 0;
}

int main(void)
{
        RT_TASK *maint;
        int thread;

        printf("GIVE THE NUMBER OF INTERRUPTS YOU WANT TO COUNT: ");
        scanf("%d", &maxcnt);

        if (!(maint = rt_task_init(nam2num("MAIN"), 1, 0, 0))) {
                printf("CANNOT INIT MAIN TASK > MAIN <\n");
                exit(1);
        }
        // create semaphore to notify main() when interrupt occurs
        if (!(dspsem = rt_sem_init(nam2num("DSPSEM"), 0))) {
                printf("CANNOT INIT SEMAPHORE > DSPSEM <\n");
                exit(1);
        }
        // ask for permission to access the parallel port from user-space
        if (iopl(3)) {
                printf("iopl err\n");
                rt_task_delete(maint);
                rt_sem_delete(dspsem);
                exit(1);
        }

        outb_p(0x10, BASEPORT + 2); //set port to interrupt mode; pins are output

        thread = rt_thread_create(timer_handler, NULL, 10000);  // create thread
        while (end) {   // wait until thread went to hard real time
                usleep(100000);
        }

        while (intcnt < maxcnt) {
                rt_sem_wait(dspsem);
                printf("unixtime = %lld\n", unixtime);
                printf("RETVAL %d, OVERRUNS %d, INTERRUPT COUNT %d\n", retval, ovr, intcnt);
        }
        end = 1;
        printf("TEST ENDS\n");
        outb_p(0, BASEPORT);
        outb_p(255, BASEPORT); // generate final interrupt to unblock rt_irq_wait
        outb_p(0, BASEPORT);

        rt_release_irq_task(PARPORT_IRQ);
        rt_thread_join(thread);
        rt_task_delete(maint);
```

```
        rt_sem_delete(dspsem);
        return 0;
}
```

Load the RTAI modules, aswell as our extension-module with "loadmods.sh":

```
#!/bin/sh
TMP=$PWD
cd /usr/realtime/modules/

insmod ./rtai_hal.ko
insmod ./rtai_lxrt.ko
insmod ./rtai_sem.ko
insmod ./rtai_usi.ko

cd $TMP

insmod ./rt_getunixtime.ko
```

Remove the modules with "rmmods.sh":

```
#!/bin/sh
rmmod rt_getunixtime
rmmod rtai_usi
rmmod rtai_sem
rmmod rtai_lxrt
rmmod rtai_hal
```

Last-Modified: Sat, 04 Feb 2006 16:13:51 GMT