29/01/2019 – Exam of Real Time Operating Systems

ACME-HEALTH decides to develop a new technology based on biosensors for monitoring patients that possibly need help.

The Donut© bio-scanning device (Fig 1) is able to scan the body of a person in a few seconds and acquire all relevant data (including temperature, blood pressure, hearth rate, etc.). In order to perform the scanning, the person lies on a platform that can move forward and backward: the Donut© requires one input signal to control a motor that moves the platform, and it returns the acquired biodata to be processed by a computer.

Notice that, depending on the value of the biodata acquired (e.g., if anomalous values are found), the scanning system can decide to slow down the scanning velocity, temporarily stop, or even to re-scan a specific area, in order to acquire higher resolution data from that part of the body. This is achieved by opportunely controlling the motor.

Figure 1. Donuts© bio-scanning device.

Once the biodata have been processed to produce a diagnosis of the health state of the patient, they are used by the Donut© to suggest a possible treatment.

Technical specifications

The Donut© device is managed by a dedicated embedded computer.

In order to control the motor allowing the Donut© device to scan patients and to acquire data, a dedicated control board is used, that receives velocity commands V from a task running on the processor and sends proper signals to the motor. Specifically, to write on the control board a value V, it is necessary to write into an 8-bit register at the address 0x260. In order to read biodata it is necessary to read from 6 contiguous 8-bit registers, starting from the address 0x300.

Finally notice also that, by opportunely programming an 8-bit control register at the address 0x360 (it is necessary to write the value 0x10), the control board is enabled to launch an interrupt on IRQ9 whenever new biodata are available in the input registers.

The onboard PC is required to perform the following tasks:

- J1 acquires biodata at the maximum frequency of 50Hz;
- J2 processes biodata as they are acquired, and checks for anomalous values; also, J2 sends a message when the whole body of the patient has been scanned.
- J3 sends proper velocity values to the motor in order to perform a standard scan of the patient body.
- J4, in the case that anomalous biodata have been detected in a specific part of the body, produces a new set of reference velocity values aimed at acquiring higher resolution data from that part of the body: these velocity values are, on their turn, sent to J3.
- J5, when the whole body has been scanned, performs more complex computations and produces a diagnosis on the basis of all the biodata acquired.

As a consequence of various statistical analysis, the computational times required for the execution of tasks J1…J5 turns out to be, in the worst case, **C1 = 2ms; C2 = 5ms; C3 = 2ms; C4 = 2ms; C5 = 100.**

Exercise 1 (9 points): **a - max score 1)** tell which tasks are to be considered periodic, which sporadic and which aperiodic. **b - max score 3)** propose a period of periodic tasks and manage opportunely sporadic tasks in order to meet i) the conditions of the general schedulability theorem and ii) sufficient conditions for RM. iii) Assign the appropriate priorities to tasks. **c - max score 5)** After an analysis of requirements, it turns out that we must add one a periodic tasks: J6 with maximum duration C6 = 5 ms. Assume to schedule the aperiodic tasks in the background, using a FIFO policy. Compute which is the maximum delay in execution (computed as termination time – arrival time) of C6 in the worst case, by performing an analysis of the idle times.

Exercise 2 (6 points): Consider three periodic tasks J1, J2, J3 ordered with descending priority (it holds C1=4, T1=10; C2=4, T2=20; C3=4, T3=40), not necessarily corresponding to the previous exercise). Task J1 and task J2 share a semaphore S1 which protects a critical region whose duration is 1ms; J2 and J3 share a semaphore S2 which protects a critical region whose duration is 1ms. J1 and J3 share a semaphore S3 which protects a critical region whose duration is 2ms. **a – max score 6**) perform again the schedulability analysis by assuming to use Priority Ceiling for accessing shared resources.

Esercize 3 (4 points): **a- max score 2)** briefly describe in 9 points (one line per point) the pros and cons of i) WindowsCE, ii) QNX, iii) VxWorks. **b – max score 2)** briefly describe in 6 points (one line per point) the pros and cons of the following mechanisms for communication / synchronization: i) FIFO queues, ii) semaphores + shared memory.

Exercise 4 (7 points): Initially, all tasks J1- J6 are implemented as standard Posix threads. Let us focus on J1 and J3, the former acquiring biodata and the latter sending proper velocity values to the motor in order to perform a scan of the patient body.

To perform read operations from the registers of the control board, a char driver is developed which is loadable as a module in runtime. Even if the driver is unique, we are requested to use two special device file "/ dev / motors " and "/ dev / sensors " to access the driver functionalities: the first is used for controlling velocities of the motors, the second for the acquisition of biodata.

a - max score 1) describe - in a few words - the shell operations needed to compile a driver, prepare the file system for its use, and load it into memory

b - max score 3) using the reference manual, write a schematic diagram of the code contained in a "mininal" driver that provides system calls for reading and writing, as well as the transition from blocking to non-blocking state (and vice versa), reporting only the prototypes of each function and commenting their use. Clarify, for each function, which event (shell commands or system calls) determines its execution.

c - max score 3) will fill properly the structure file_operations and show how it is possible to ensure that, in the event that it is necessary, the two special device files "/ dev / motors " and "/ dev / sensors " can be associated with two different sets of system calls.

Exercise 5 (9 points): In a second phase, it is decided that J1, and J3 are executed as RTAI tasks, whereas the remaining tasks are executed in user space as POSIX threads.

a - max score 3) using the reference manual, write a schematic diagram of the code contained in a "minimal" RTAI module, so that it is indicated by appropriate comments, i) where tasks and the system timer are initialized; ii) where is the code specific to each task.

b - max score 3) write in detail in the appropriate position, i) functions to initialize the timer in periodic mode, ii) functions to initialize the tasks and make them periodic with an EDF scheduling policy iii) the internal structure of the code of J1 and J3, including I/O operations required to send commands to actuators by writing in the proper output registers. Use comments in natural language ("the task here does this ...") to highlight the application-specific code of each task.

c - max score 3) Implement the mechanism allowing J4 to send a new set of velocities to J3, i.e., whenever anomalous biodata have been detected and a specific part of the body needs to be scanned at a higher resolution. In particular, show how data can be sent from user space to kernel space using a FIFO queue with a proper FIFO handler.

cycles_t get_cycles(void);
int rt_free_global_irq (unsigned int irq);
int rt_request_global_irq (unsigned int irq, void(*handler)(void));
int rt_sem_wait_timed (SEM *sem, RTIME delay)int rt_sem_wait_until (SEM *sem, RTIME time)
int (*fsync) (struct file *file, struct dentry *dentry, int datasync);
int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
int (*ioctl) (struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg);
int (*mmap) (struct file *, struct vm_area_struct *);
int (*open) (struct inode *, struct file *);
int (*release) (struct inode *, struct file *);
int access_ok(int type, const void *addr, unsigned long size);
int check_mem_region(unsigned long start, unsigned long len);
int close(int fd);
int gettimeofday(struct timeval *tv, struct timezone *tz);
int ioperm(unsigned long *from*, unsigned long *num*, int *turn_on*);
int iopl(int *level*);
int nanosleep(const struct timespec **req*, struct timespec **rem*);
int open(const char *pathname, int flags);
int printk(const char*fmt, ...)
int probe_irq_off(unsigned long);
int pthread_attr_init(pthread_attr_t *attr)
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
int pthread_attr_setinheritsched(pthread_attr_t *attr, int inherit);
int pthread_attr_setschedparam(pthread_attr_t *attr, const struct sched_param *param);
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
int pthread_attr_setscope(pthread_attr_t *attr, int scope);
int pthread_cancel_(pthread t THREAD) ;
int pthread_cond wait (pthread cond t *COND, pthread mutex t *MUTEX) ;
int pthread_cond_broadcast (pthread cond t *COND) ;
int pthread_cond_destroy (pthread cond t *COND) ;
int pthread_cond_init (pthread cond t *COND, pthread condattr t *cond ATTR) ;
int pthread_cond_signal (pthread cond t *COND) ;
int pthread_cond_timedwait (pthread cond t *COND, pthread mutex t *MUTEX, const struct timespec *ABSTIME) ;
int pthread_create(pthread_t * thread, const pthread_attr_t * attr, void * (*start_routine)(void *), void *arg);
int pthread_detach(pthread_t th);
int pthread_join (pthread t TH, void **thread RETURN) ;
int pthread_mutex_destroy (pthread mutex t *MUTEX);
int pthread_mutex_init (pthread mutex t *MUTEX, const pthread mutexattr t *MUTEXATTR) ;
int pthread_mutex_lock (pthread mutex t *mutex)) ;
int pthread_mutex_timedlock (pthread mutex t *MUTEX, const struct timespec *ABSTIME) ;
int pthread_mutex_trylock (pthread mutex t *MUTEX) ;
int pthread_mutex_unlock (pthread mutex t *MUTEX);
int pthread_setschedparam(pthread_t target_thread, int policy, const struct sched_param *param);
int register_chrdev(unsigned int major, const char *name, struct file_operations *fops);
int request_irq(unsigned int irq, void (*gest)(int, void *, struct pt_regs *), unsigned long flags, const char *devname, void *dev_id);
int rt_get_task_state (RT_TASK **task*);
int rt_sem_delete (SEM *sem)
int rt_sem_signal (SEM *sem)
int rt_sem_wait (SEM *sem)
int rt_task_delete (RT_TASK **task*);
int rt_task_make_periodic(RT_TASK *task, RTIME start_time, RTIME period);
int rt_task_make_periodic_relative_ns (RT_TASK **task*, RTIME *start_delay*, RTIME *period*);
int rt_task_resume (RT_TASK **task*);
int rt_task_suspend (RT_TASK **task*);
int rt_task_use_fpu (RT_TASK* *task*, int *use_fpu_flag*);
int rtf_create (unsigned int *fifo*, int *size*);
int rtf_create_handler (unsigned int *fifo*, int (**handler*)(unsigned int *fifo*));

int rtf_destroy (unsigned int *fifo*);
int rtf_get (unsigned int *fifo*, void **buf*, int *count*);
int rtf_put (unsigned int *fifo*, void **buf*, int *count*);
int rtf_reset (unsigned int *fifo*);
int settimeofday(const struct timeval *tv , const struct timezone *tz);
int setuid(uid_t uid);
int unregister_chrdev(unsigned int major, const char *name);
int wait_event_interruptible(wait_queue_head_t queue, int condition);
loff_t (*llseek) (struct file *, loff_t, int);
MAJOR(kdev_t dev);
MINOR(kdev_t dev);
MOD_DEC_USE_COUNT (decrementa)
MOD_IN_USE restituisce true se il contatore non è uguale a 0
MOD_INC_USE_COUNT (incrementa)
PTHREAD ADAPTIVE MUTEX INITIALIZER NP,
PTHREAD ERRORCHECK MUTEX INITIALIZER NP
PTHREAD MUTEX INITIALIZER,
PTHREAD RECURSIVE MUTEX INITIALIZER NP,
rdtsc(low,high);
rdtscl(low);
RT_TASK *rt_whoami (void);
rt_task_init(RT_TASK *task, void *rt_thread, int data, int stack_size, int priority, int uses_fp, void *sig_handler);
RTIME nano2counts(int nanoseconds);
RTIME rt_get_cpu_time_ns (void);
RTIME rt_get_time (void);
RTIME rt_get_time_ns (void);
RTIME start_rt_timer(RTIME period);
SEM sem;
ssize_t (*read) (struct file *, char *, size_t, loff_t *);
ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
ssize_t read(int fd, void *buf, size_t count);
ssize_t read(int fd, void *buf, size_t count);
unsigned rt_startup_irq (unsigned irq);
unsigned inb(unsigned port);
unsigned inl(unsigned port);
unsigned int (*poll) (struct file *, poll_table *);
unsigned inw(unsigned port);
unsigned long copy_from_user(void *to, const void *from, unsigned long count);
unsigned long copy_to_user(void *to, const void *from, unsigned long count);
unsigned long probe_irq_on(void);
unsigned readb(address);
unsigned readl(address);
unsigned readw(address);
void rt_disable_irq (unsigned irq);
void rt_enable_irq (unsigned irq);
void rt_shutdown_irq (unsigned irq);
void, rt_spv_RMS (int cpuid)
void rt_task_set_resume_end_times (RTIME resume_time, RTIME end_time)
void (*handler)(int, void *, struct pt_regs *)
void *ioremap(unsigned long phys_addr, unsigned long size);
void *iounmap (void *addr)
void *kmalloc (size_t size, int prio);
void *rtai_kmalloc(unsigned long name, int size); void *rtai_malloc (unsigned long name, int size);
void barrier(void)
void disable_irq(int irq);
void disable_irq_nosync(int irq);
void do_gettimeofday(struct timeval *tv);
void enable_irq(int irq);
void init_waitqueue_head (wait_queue_head_t *queue);

void interruptible_sleep_on(wait_queue_head_t *queue);
void interruptible_sleep_on_timeout(wait_queue_head_t *queue, long timeout);
void kfree (const void *ptr);
void mb(void);
void mdelay(unsigned long msecs);
void memcpy_fromio(unsigned long, unsigned long, unsigned int count);
void memcpy_toio(unsigned long, unsigned long, unsigned int count);
void memset_io(unsigned long, char fill, int count)
void outb(unsigned char byte, unsigned port);
void outl(unsigned longword, unsigned port);
void outw(unsigned short word, unsigned port);
void pthread cancel (pthread t THREAD) ;
void pthread exit (void *RETVAL) ;
void pthread_exit(void *retval);
void release_mem_region (unsigned long start, unsigned long len);
void request_mem_region(unsigned long start, unsigned long len, char * name);
void rmb(void);
void rt_busy_sleep (int *nanosecs*);
void rt_linux_use_fpu (int *use_fpu_flag*);
void rt_sem_init (SEM *sem, int value)
void rt_set_periodic_mode(void);
void rt_sleep (RTIME *delay*);
void rt_sleep_until (RTIME *time*);
void rt_task_signal_handler (RT_TASK **task*, void (**handler*)(void));
void rt_task_wait_period(void);
void rt_task_yield (void);
void rtai_free (int name, void *adr)
void rtai_kfree (int name)
void sleep_on(wait_queue_head_t *queue);
void sleep_on_timeout(wait_queue_head_t *queue, long timeout);
void spin_lock(spinlock_t *lock);
void spin_lock_init(spinlock_t *lock);
void spin_lock_irqsave(spinlock_t *lock, unsigned long flags);
void spin_unlock(spinlock_t *lock);
void spin_unlock_irqrestore(spinlock_t *lock, unsigned long flags);
void udelay(unsigned long usecs);
void wait_event(wait_queue_head_t queue, int condition);
void wake_up(wait_queue_head_t *queue);
void wake_up_interruptible(wait_queue_head_t *queue);
void wake_up_interruptible_sync(wait_queue_head_t *queue);
void wake_up_sync(wait_queue_head_t *queue);
void wmb(void);
void writeb(unsigned value, address);
void writel(unsigned value, address);
void writew(unsigned value, address);
void free_irq(unsigned int irq, void *dev_id)