Computer System Engineering II (DT4013)

Real-Time Micro-Kernel Project

Essential Global Variables and Project Properties

1 Required Global Variables:

```
int Ticks; /* global sysTick counter */
int KernelMode; /* can be equal to either INIT or RUNNING (constants defined
                 * in "kernel_functions.h")*/
TCB *PreviousTask, *NextTask; /* Pointers to previous and next running tasks */
list *ReadyList, *WaitingList, *TimerList;
```

2 The `main` function for testing your kernel

```
#include "system_sam3x.h"
#include "at91sam3x8.h"
#include "kernel_functions.h"

void main (void)
{
   SystemInit();
   SysTick_Config(100000);
   SCB->SHP[((uint32_t)(SysTick_IRQn) & 0xF)-4] = (0xE0);
   isr_off(); // Disable Interrupts

   init_kernel(); // Kernel initialization function you have to implement as a part of Task Administration

  /*
    your code to test the implemented kernel
  */

  run(); /* starts the kernel & enables interrupts you have to implement as a
          * partof Task Administration
          */
}
```

3 The API header file

The header file **"kernel_functions.h"** contains the definition of:

- The *Task Control Block (TCB)* is a structure (`struct`) which represents the context of a task. The TCB contains the stack pointer, and the 8 registers (R4 to R11) which are regularly saved and stored in the TCB fields dedicated to registers. The remaining registers (R0 to R3, R12, LR, PC, PSR) are saved and retrieved from the stack by the hardware when:
  a) you call the assembly function `SwitchContext()`, or,
  b) you call the assembly functions `isr_off()`, `isr_on()` or,
  c) a SysTick interrupt (`SysTick_Handler()`) is serviced.

- The `list` is a structure that refers to the double linked lists `ReadyList`, `WaitingList` and `TimerList`.
- The *list object* (`listobj, l_obj`) is a structure that represents the nodes composing the double linked lists `ReadyList`, `WaitingList` and `TimerList`.
- `mailbox` is a structure that represent the double linked list for inter-process communication between running tasks on the kernel. `*pHead`, `*pTail` are pointers for the first and last message in a mailbox, respectively. `nDataSize` refers to the size of the message in bytes. `nMaxMessages` refers to the maximum capacity of a mailbox. `nMessages` refers to the actual number of messages in a mailbox. `nBlockedMsg` refers to the number of blocking messages in a mailbox.
- `msg` is a structure that refers to a single message. `*pData` is a pointer for the actual message. `Status` determines if the message from a `SENDER` or `RECEIVER`. `*pBlock` is a pointer that refers to the blocked task.
- Constants: There exist a set of constants that are used identify certain events happening during the execution of the kernel such as, `FAIL`, `OK`, `SENDER`, `RECEIVER`, `NOT_EMPTY`,…etc.

4 Initializing TCB

The initializing of the TCB happens only in `create_task()` function. The following code shows how to do it:

```
exception create_task (void (*taskBody)(), unsigned int deadline)
{
   TCB *new_tcb;
   new_tcb = (TCB *) calloc (1, sizeof(TCB));
   /* you must check if calloc was successful or not! */

   new_tcb->PC = taskBody;
   new_tcb->SPSR = 0x21000000;
   new_tcb->Deadline = deadline;

   new_tcb->StackSeg [STACK_SIZE - 2] = 0x21000000;
   new_tcb->StackSeg [STACK_SIZE - 3] = (unsigned int) taskBody;
   new_tcb->SP = &(new_tcb->StackSeg [STACK_SIZE - 9]);
   // after the mandatory initialization you can implement the rest of the
suggested pseudocode
}
```

5 Context Switching

There exist three implemented functions that do context switching. They are:

- `LoadContext_In_Run()` in the function `run()`
- `LoadContext_In_Terminate()` in the function `terminate()`
- `SwitchContext()` every where else (e.g. `create_task()`)
  Hint : Keep in mind that `SwitchContext()` enables interrupts towards the end of its execution !

6 Required format for `run()`

```
void run (void)
{
   NextTask = ReadyList->pHead->pTask;

   LoadContext_In_Run();

   /* supplied to you in the assembly file
    * does not save any of the registers
    * but simply restores registers from saved values
    * from the TCB of NextTask
    */
}
```

7 Required format for `terminate()`

```
void terminate (void)
{
   isr_off();
   leavingObj = extract(ReadyList->pHead);
   /* extract() detaches the head node from the ReadyList and
    * returns the list object of the running task */
   NextTask = ReadyList->pHead->pTask;
   switch_to_stack_of_next_stack();


   free(leavingObj->pTask);
   free(leavingObj);
   LoadContext_In_Terminate();
    /* supplied to you in the assembly file
     * does not save any of the registers. Specifically, does not save the
     * process stack pointer (psp), but
     * simply restores registers from saved values from the TCB of NextTask
     * note: the stack pointer is restored from NextTask->SP
     */
}
```

8 Use memcpy to copy data in the communication functions

In communication functions (e.g. `send_wait()`), you have to copy data from the sender's area to the receiver's area. To do so in the mailbox called `mBox`, say:

```
memcpy( receiving_pointer, sending_pointer , mBox->nDataSize)
```

Please see http://www.cplusplus.com/reference/cstring/memcpy/ for more details.

9 Avoid using `printf` to debug

If you must use `printf`, then make sure that interrupts are disabled before calling it.

Do not forget enable interrupts back again after using `printf`.