HÖGSKOLAN I HALMSTAD

Computer System Engineering II (DT4013)

Real-Time Micro-Kernel Project

Lab 1: Task Administration

Function Description	Suggested Pseudocode
exception init_kernel () This function initializes the kernel and its data structures (**ReadyList**, **WaitingList**, **TimerList which are mentioned in the project files**) and leaves the kernel in start-up mode. The **init_kernel** call must be made before any other call is made to the kernel. **Argument** None **Return parameter** **exception**: Description of the function's status, i.e. **FAIL** / **OK** (**exception** values defined as constants in **kernel.h**).	Set tick counter to zero Create necessary data structures Create an **Idle task** Set the kernel in INIT mode Return status

exception create_task(void(*task_body)(), uint deadline)
 	
This function creates a task. If the call is made in start- up mode, i.e. the kernel is not running, only the necessary data structures will be created. However, if the call is made in running mode, it will lead to a rescheduling and possibly a context switch.
 	
Argument
***task_body:** A pointer to the C function holding the code of the task.
 	
deadline: The kernel will try to schedule the task so it will meet this deadline.
 	
Return parameter
exception: Description of the function's status, i.e. **FAIL** / **OK** (**exception** values defined as constants in **kernel.h**).
	Allocate memory for TCB
	Set deadline in TCB
	Set the TCB's PC to point to the task body Set up stack frame and set TCB's SP to point to the correct cell in stack segment
	
	IF KernelMode == INIT **THEN**
	Insert new task in ReadyList
	Return status
	ELSE
	Disable interrupts
	Update PreviousTask
	Insert new task in ReadyList
	Update NextTask
	Switch context
	ENDIF
	Return status

void run() This function starts the kernel and thus the system of created tasks. It will leave control to the task with	Initialize interrupt timer {Ticks = 0;} Set **KernelMode** = RUNNING

tightest deadline. Therefore, it must be placed last in the application initialization code. **Note:** The C function **LoadContext_In_Run()** enables interrupts. So there is no need to call **isr_on()**.	Set **NextTask** to equal TCB of the task to be loaded Load context using: { **LoadContext_In_Run()**; }
void terminate() This call will terminate the running task. All data structures for the task shall be removed. Before removing the data structure, make sure to switch to the process stack of the task to be called. Thereafter, another task will be scheduled for execution.	Disable interrupts Remove running task from **ReadyList** Set **NextTask** to equal TCB of the next task Switch to process stack of task to be loaded **{ switch_to_stack_of_next_task(); }** Remove data structures of task being terminated Load context using: **{LoadContext_In_Terminate();}**