Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Project in Embedded Systems - 7.5 ECTS

API DESIGN SUGGESTION TEAM RUBUS

Afram Afrem
aam20002@student.mdh.se

Lukas Dust
ldt20001@student.mdh.se

Victor Nicholas Ebirim
vem20001@student.mdh.se

Andreas Mäkilä
ama17002@student.mdh.se

Aymen Nouidha
ana20001@student.mdh.se

2020-11-20

Contents

1. Scheduler (Aymen Nouidha) 1

1.1. StartSchedular . . . 1

1.2. TaskQueueAssignement . . . 1

2. Tasks (Victor Ebirim) 1

2.1. ROSA_tcbKill . . . 1

2.2. ROSA_start (modified) . . . 1

3. Clock (Lukas Dust, Afram Afrem) 2

3.1. Directly required by the Customer . . . 2

3.1..1 ROSA_sysTickWait . . . 2

3.1..2 ROSA_sysTickWaitUntil . . . 2

3.2. Optional / Not directly mentioned . . . 2

3.2..1 ROSA_getTickCount . . . 2

3.2..2 ROSA_sysTickWaitTask . . . 3

3.2..3 ROSA_sysTickWaitTaskUntil . . . 3

4. Semaphores (Andreas Mäkilä) 3

4.1. ROSA_sem . . . 3

4.2. ROSA_semCreate . . . 3

4.3. ROSA_semTake . . . 4

4.4. ROSA_semGive . . . 4

1. Scheduler (Aymen Nouidha)

1.1. StartSchedular

Prototype: Void StartSchedular(Void);
Description: Starts the OS scheduler
Parameters: No parameters
Return: Void
Motivation: It is the cornerstone of the OS scheduling capabilities, it initializes and starts the scheduler.

1.2. TaskQueueAssignement

Prototype: bool TaskQueueAssignement(TCB* , Queue*);
Description: Assign a task to a defined queue and insert it in the proper order depending on its priority
Parameters: TCB* is a pointer to the task structure, Queue* is a pointer to the first item of the queue linked List
Return: bool, it return **True** , if the operation was carried on successfully or **False** otherwise
Motivation: The scheduler needs this function to take tasks from the Blocked queue to the ready queue whenever a new period for the said tasks begin, and it needs to be inserted in the proper order.

2. Tasks (Victor Ebirim)

2.1. ROSA_tcbKill

Prototype: unsigned char ROSA_tcbKill(tcb *TCB);
Description: Removes a task from kernel
Parameters: tcb *TCB
- a pointer to the TCB block to be deleted
Return: unsigned int
–0: Successful
–1: Unsuccessful
Motivation: Required by the customer

2.2. ROSA_start (modified)

Prototype: void ROSA_start(tcb *TCB, char *id, void *taskFunc, unsigned char taskPrio, int *stack, int stackSize);
Description: Creates a TCB entry according to the given parameters
Parameters: tcb *TCB
- A pointer to the TCB block to be created
char *id
- A identification for the TCB block of length NAMESIZE (default NAMESIZE = 4)

void *taskFunc
- A pointer to the function which are to be executed by the task
unsigned char taskPrio
- The task priority
int *stack
- A pointer to the task stack area
int stackSize
- The maximum allowed stack for this task

Return: None
Motivation: Includes additional parameter Task priority

3. Clock (Lukas Dust, Afram Afrem)

3.1. Directly required by the Customer

3.1..1 ROSA_sysTickWait

Prototype: void ROSA_sysTickWait(int waitTicks);
Description: Suspends the using Task for the given amount of ticks
Parameters: int waitTicks
- amount of Ticks which the task should be suspended

Return: void
Motivation: Required by the customer

3.1..2 ROSA_sysTickWaitUntil

Prototype: void ROSA_sysTickWaitUntil(int tickTime);
Description: Suspends the using Task until the stated system Tick time
Parameters: int tickTime
- Absolute time in ticks till when the task should be suspended

Return: void
Motivation: Required by the customer

3.2. Optional / Not directly mentioned

3.2..1 ROSA_getTickCount

Prototype: int ROSA_getTickCount();
Description: Returns the value of the Tickcounter which keeps track of the system time
Parameters: NONE
Return: int
- actual value of the system tick count

Motivation: This function is not directly mentioned by the customer, but it can help the user to have the opportunity to get the actual system time. E.g. in the situation for using it to define the end of the absolute delay.

3.2..2 ROSA_sysTickWaitTask

Prototype: void ROSA_sysTickWaitTask(int waitTicks, tcb *task);
Description: Suspends a specified Task for the given amount of ticks
Parameters: int waitTicks
- amount of Ticks which the task should be suspended
tcb *task
- a pointer to the task structure
Return: void
Motivation: This functionality is not mentioned by the customer, but could be useful to the user to suspend specific Tasks during the execution of another one. E.g. in the case that after the execution of an Task an other one does not need to execute anymore in that time. An implementation will be considered, after asking the customer.

3.2..3 ROSA_sysTickWaitTaskUntil

Prototype: void ROSA_sysTickWaitUntil(int tickTime, tcb *task);
Description: Suspends a specified Task until the stated system Tick time
Parameters: int tickTime
- Absolute time in ticks till when the task should be suspended
tcb *task
- a pointer to the task structure
Return: void
Motivation: This functionality is not mentioned by the customer, but could be useful to the user to suspend specific Tasks during the execution of another one. E.g. in the case that after the execution of an Task an other one does not need to execute anymore in that time. An implementation will be considered, after asking the customer.

4. Semaphores (Andreas Mäkilä)

4.1. ROSA_sem

Prototype: typedef struct{bool isFree; int ceilPrio; int oldPrio;}ROSA_sem;
Description: The struct used to represent a semaphore.
Elements: bool isFree
- False if semaphore is locked. Initialize to true.
int ceilPrio
- Used for IPCP. Initialize to semaphore's highest prio task.
int oldPrio
- Used for IPCP to store a task's normal priority.
Motivation: The semaphore was supposed to be binary and I wanted to keep it as small as possible. This was the most efficient struct I could come up with for that purpose.

4.2. ROSA_semCreate

Prototype: int ROSA_semCreate(tcb *task, int listLength);
Description: Function to create semaphores.
Parameters: tcb *task
- The list of tasks that will use the semaphore.
int listLength
- The length of the task list
Return: ROSA_sem
- The created semaphore
Motivation: This create function makes the act of creating a semaphore easier than having to manually construct a ROSA_sem variable.

4.3. ROSA_semTake

Prototype: int ROSA_semTake(ROSA_sem *semphor, tcb *task);
Description: Function which "takes" (or "locks") a semaphore.
Parameters: ROSA_sem *semphor
- A pointer to the semaphore to be taken.
tcb *task
- A pointer to the TCB task doing the locking.
Return: int
–0: Successful
–1: Unsuccessful (due to timeouts)
Motivation: Required to be able to use the semaphores

4.4. ROSA_semGive

Prototype: int ROSA_semGive(ROSA_sem *semphor, tcb *task);
Description: Function that "gives" (or "unlocks") a semaphore.
Parameters: ROSA_sem *semphor
- A pointer to the semaphore to be given back.
tcb *task
- A pointer to the TCB task doing the unlocking.
Return: int
–0: Successful
–1: Unsuccessful (due to semaphore already being free)
Motivation: Required to be able to use the semaphores