Implementación de prioridades y estados de tarea con base de tiempo Timer 1 en placa Nucleo-F429

Alumno: Carlos Herrera

Estado de tareas

```
static void TaskFSMstate(osTaskStatusType *state, bool_t wait, bool_t terminate)
{
	switch (*state) {
		case OS_TASK_READY:
			*state=OS_TASK_RUNNING;
			break;
		case OS_TASK_RUNNING:
			*state=OS_TASK_READY;
			if (wait) {
				*state=OS_TASK_BLOCK;
				break;
			}
			if (terminate) {
				*state=OS_TASK_SUSPEND;
				break;
			}
			break;
		case OS_TASK_BLOCK:
			*state=OS_TASK_READY;
			break;
		case OS_TASK_SUSPEND:
			*state=OS_TASK_READY;
			break;
		default:
			break;
	}
}
```

Tarea Idle

```
/* Tarea Idle */
static void CreateIdleTask(void)
{
    osTaskCreate(&Idle_Task, OS_LOW_PRIORITY+1, osIdleTask);
}
```

En el scheduler:

```
				// Revisa si todas las tareas estan en estado bloqueado
				if (osKernel.priority_matrix[var][var2]->state != OS_TASK_BLOCK)
				{
					osKernel.idle_flag=false;
				}
```

```
	// Habilita el Idle Task si las tareas se encuentran bloqueadas
	if (osKernel.idle_flag) {
		osKernel.nextTask  = &Idle_Task;
	}
```

Si se llama a la api “void osDelay(const uint32_t tick)” no se ejecuta

```
	running_task=getRunningTask();
	// No se ejecuta si la tarea en Running es la tarea Idle
	if (running_task!=&Idle_Task) {
	    if (!running_task->active_delay) {
		running_task->active_delay=true;
		running_task->time_delay=tick;
			TaskFSMstate(&(running_task->state), true, false);
		}
	}
```

API Delay

Se añade campos en la estructura de tarea.

```
    bool_t                          active_delay;
    uint32_t                        time_delay;
}osTaskObject;
```

Función osDelay

```
void osDelay(const uint32_t tick)
{
        osTaskObject* running_task;

        // Error si se encuentra en ISR
        if ((SCB->ICSR & SCB_ICSR_VECTACTIVE_Msk)!=0) {
                osErrorHook(NULL);
        }

        running_task=getRunningTask();
        // No se ejecuta si la tarea en Running es la tarea Idle
        if (running_task!=&Idle_Task) {
            if (!running_task->active_delay) {
                running_task->active_delay=true;
                running_task->time_delay=tick;
                        TaskFSMstate(&(running_task->state), true, false);
                }
        }
}
```

Actualizo la cuenta del tick a través del scheduler.

```
        updateCountersDelays();
```

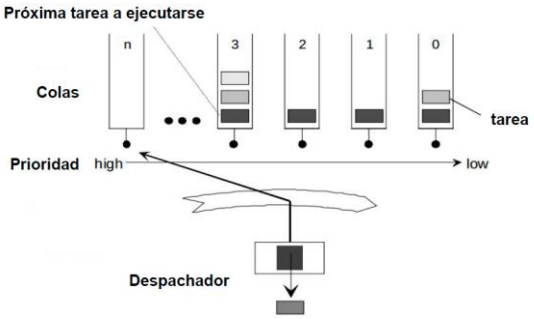
```
static void updateCountersDelays(void)
{

        for (int var = 0; var < MAX_NUMBER_PRIORITY; ++var) {
                for (int var2a = 0; var2a < MAX_NUMBER_TASK; ++var2a) {
                        if (osKernel.priority_matrix[var][var2a]!=NULL && osKernel.priority_matrix[var][var2a]->active_delay &&
osKernel.priority_matrix[var][var2a]->time_delay>0)
                        {
                                osKernel.priority_matrix[var][var2a]->time_delay--;
                        }
                        if (osKernel.priority_matrix[var][var2a]->time_delay==0 && osKernel.priority_matrix[var][var2a]-
>state==OS_TASK_BLOCK) {
                       TaskFSMstate(&(osKernel.priority_matrix[var][var2a]->state), false, false);
                       osKernel.priority_matrix[var][var2a]->active_delay=false;
                        }
                }
        }
}
```

Nota: Requiere mejorar el algoritmo de Delay, debido a que se presentan diferencias visibles al delay requerido.

Prioridades

Se creo una matriz de niveles de prioridad vs número de tareas y se hace un recorrido desde la prioridad más alta hacia la más baja buscando la tarea en estado Ready, así mismo, se guarda el último índice de tareas donde se encontró a fin de continuar siendo una ejecución circular.




```
    bool_t endfind;
    bool_t least_one;

    least_one=false;
         osKernel.idle_flag=true;
         endfind=false;

         for (int var = 0; var < MAX_NUMBER_PRIORITY; ++var) {

                  for (int var2a = osKernel.priority_startindex[var]; var2a < MAX_NUMBER_TASK + osKernel.priority_startindex[var];
++var2a) {
                           int var2;
                           var2=var2a%MAX_NUMBER_TASK;

                           if (osKernel.priority_matrix[var][var2]!=NULL)
                           {
                                    // Revisa si todas las tareas estan en estado bloqueado
                                    if (osKernel.priority_matrix[var][var2]->state != OS_TASK_BLOCK)
                                    {
                                             osKernel.idle_flag=false;
                                    }

                                    // Busca que haya al menos una tarea en estado running
                                    // en caso solo exista una tarea en un nivel de prioridad
                                    if (osKernel.priority_matrix[var][var2]->state == OS_TASK_RUNNING)
                                    {
                                             least_one=least_one | true;
                                    }

                                    if (osKernel.priority_matrix[var][var2]->state == OS_TASK_READY  && !endfind) {
                                             osKernel.nextTask=osKernel.priority_matrix[var][var2];
                                             osKernel.priority_startindex[var]=var2;
                                             least_one=least_one | true;
                                             endfind=true;
                                    }
                           }
                  }
                  if (least_one) {
                           break;
                  }
         }
```