List management – data structures and functions

Much of the administration in a real-time kernel consists of list management. In this initial step, three different lists are created. These, plus their elements have predefined names which some are described below. It is important to the below-defined structures, etc. use the given name in your program, too.

Overview description of the task

Create three linked lists (Timer List, Waiting List and Ready List).

- Create functions that can input and extract elements from these lists.
- Test the lists and see that you can handle various sizes of lists. "From zero to many elements."
- Remove list items from a list and put in another.

Features

Three lists will be created: **Timer List**, **Waiting List** and **Ready List**. In this initial stage, we will not use all parts of the list items. Depending on the listing type, the different types of structure to be used lists are also sorted according to their individual principle (why this is the case is described in the lectures).

Timer List:

• The insertion of an element into the list, pass an integer with the function call. Insertion in the list is done so that the element with the lowest value of nTCnt appears first.
• Extraction is always done from the front, i.e. at the "head-element".

Waiting List:

• A list, where the list is sorted so that the element with the lowest value of TCB-> Deadline is first.
• Extraction made by the use of a pointer to the list element, `struct l_obj * pBlock`

Ready List

• The element with the lowest value of TCB-> Deadline is first placed first in the list.
• Extraction is always done from the front, i.e. at the "head-element".

The lists can be of type:

```
/ / Generic list
typedef struct (
listobj * pHead;
listobj * pTail;
) List;
```

List-elements

```
struct l_obj; /* Forward declaration*/
/* Generic list item*/
typedef struct l_obj {
           TCB *pTask;
           unsigned int nTCnt;
           msg *pMessage;
           struct l_obj *pPrevious;
           struct l_obj *pNext;
} listobj;
```

Each list item is also a pointer to a so-called TCB-blocks. This is defined as:

```
// Task Control Block, TCB
typedef struct
{
           void (*PC)();
           unsigned int *SP;
           unsigned int Context[CONTEXT_SIZE];
           unsigned int StackSeg[STACK_SIZE];
           unsigned int DeadLine;
} TCB;
```

As you can see, there are at this stage of the program several variables that are not used! We do not care about them at the moment. The use of these will be easier to comprehend a bit later in the course.