Real-Time Operating Systems

ESD 813

Assignment 1

(Chandramouleeswaran Sankaran)

Mouli.sankaran@iiitb.ac.in

Date: 25 Aug 2015

Date of Submission: 7th Sep 15

Assignment 1

< Course: RTOS, ESD 813>

Write C programs for each of the problems below, following the **GNU coding standards**.

Prepare the following documents too.

a. Design document explaining the algorithm and data structures
b. Test plan document giving the set of test cases the program was tested with.

1. Implement the following memory management functions for storage allocation on a heap by allocating an array on a global space. Write sample programs (test cases) which test these functions rigorously.

The order and contiguity of storage allocated by successive calls to the ***my_calloc***, ***my_malloc***, and ***my_realloc*** functions is unspecified. The pointer returned if the allocation succeeds is suitably aligned so that it may be assigned to a pointer to any type of object and then used to access such an object or an array of such objects in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint from any other object.

The pointer returned points to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer is returned. If the size of the space requested is zero, a **NULL** pointer is returned; the value returned shall be either a null pointer or a unique pointer. The value of a pointer that refers to freed space is indeterminate.

a. ***void * my*_calloc** (***size_t*** nmemb, ***size_t*** size);

   This function allocates space for an array of **nmemb** objects, each of whose size is **size**. The space is initialized to all bits zero.

b. ***void*** **my_free** (***void* *** ptr);

   This function causes the space pointed to by **ptr** to be deallocated, that is, made available for further allocation. If **ptr** is a null pointer, no action occurs. If the argument does not match with a pointer value that was returned earlier by the ***my_calloc***. ***my_malloc***, or ***my_realloc*** function, or if the space has been deallocared by a call to ***free*** or ***realloc***, the behavior is undefined.

c. ***void * my*_malloc**(**size_t** size);

   This function allocates space for an object whose size is specified by **size** and whose value is indeterminate.

d. ***void * my*_realloc** (***void **** ptr, ***size_t*** size);

   The function changes the size of the object pointed to by **ptr** to the size specified by **size**. The contents of the object shall be unchanged up to the lesser of the new and old sizes. If the new size is larger, the value of the newly allocated portion of the object is indeterminate. If **ptr** is a null pointer, the ***my_realloc*** function behaves like the ***my_malloc*** function for the specified size. Otherwise, if **ptr** does not match a pointer earlier returned by the ***my_calloc***. ***my_malloc***, or ***my_realloc*** function, or if the space has been deallocated by a call to the ***my_free*** or ***my_realloc*** function, the behavior is undefined. If the space cannot be allocated, the object pointed to by **ptr** is unchanged. If size is zero and **ptr** is not a null pointer, the object it points to is freed.

e. ***size_t*** **free_space_in_my_heap *(void*)**;

   This function returns the total size of free space available in the heap.

f. ***void*** **deframent_my_heap *(void*)**;

   This function combines all the allocated memory chunks into contiguous space without losing any of the values stored in them.

2. Implement the following functions in C, based on the **five bit pattern (**"**11011**"). Make sure proper error checks are done, based on the size of heap space made available for these functions.

   a. ***char*** ***check_bit_pattern** (***char*** *** start_addr**);

      This function checks whether the **binary bit pattern** given is found in the memory from the start address and returns the address where the pattern check has failed.

   b. ***void*** **fill_pattern** (***char*** *** start_addr**, ***char**** **end_addr**);

      This function fills the **binary bit pattern** mentioned in the static array starting from the start address till the end address.

   c. ***void*** **init_zero** (***char*** *** start_addr**, ***char**** **end_addr**);

      This function from the starting address to the end address fills the memory with **zeros**.