Malloc Implementation

Due: May 2nd, 2007
(Grading will be in room 013, CSE Resource Center from 1 to 4 pm)

In this assignment, you will create your own version of *malloc* and *free* routines that replace the standard-C library version. The program that will be used to test your routines is available on the class website. The program spawns four tasks:

1. *initialize_task* (priority level 6) creates three additional tasks, 2 to perform memory allocations and one to use the allocated information. After these three tasks are created, *initialize_task* terminates itself.
2. *free_task* (priority level 7) uses and frees half of the total allocated nodes then suspends itself for a short interval (currently set to 35 seconds). Note that when it is run for the first time, it simply initializes the turn variable and then suspends itself for 50 seconds.
3. *malloc_task1* and *malloc_task2* (priority level 8 and 9) allocate 12000 nodes each. For each task, its main responsibility is to finish the allocation of nodes before *free_task* wakes up. If this requirement is not satisfied, the program fails.

You must make sure that your *malloc* routine will allow ***malloc_task1*** **and** ***malloc_task2*** **to accomplish its responsibility in the given amount of time or shorter (50 seconds initially and 35 seconds afterward)**. You must also make sure that your *free* does not increase the execution time of *free_task* by more than 10% of the current execution time. You can use *alt_timestamp_start()* and *alt_timestamp_stop()*, the same functions used in your instruction extension assignment, to time the *free_task* function.

Steps:

1. Your work must be performed on the Altera DE-2 board. Make sure that you can get the provided image to work ASAP.

2. You can design your *malloc* and *free* using multiple tasks. However, the priority of these tasks must not be higher (lower priority numbers) than the given tasks. Also do not use priority 10. This means that your own tasks should start at priority number 11 and beyond. You are allowed to optimize your routines to match the allocation behaviors.

3. Modify *malloc_task1* so that its priority changes between 8 and 10 each time it runs. To change priority, you can use *OSTaskChangePrio*(). You must make sure that the provided test program does not starve (i.e. *free_task* tries to access nodes when the *number_of_alloc_node1* or *number_of_alloc_node2* is zero). This may mean that you have to modify *malloc_task2* to prevent starvation. You should change priority **after** or **instead of** the 80 ms delay in *task_1*. Do your routines still use the same amount of time to meet the deadlines? If your answer is yes, explain how your routine can accommodate such a change. If your answer is no, modify your routines so that they would perform well in both scenarios.

Submission:

We will grade the assignment in room 20 on Wednesday May 2nd. The grading time will be from 1:00 to 4:00 pm. After the grading period, you must submit the following programs through handin by 11:59 pm on the same day.

- *malloc_assignment.c* and *malloc_assignment_modified.c* (the modified version is the one with priority changing of malloc_task1).
- Your malloc and free routines.
- Any header files that you have created to support this assignment.
- A short report answer the following questions:
    - What are some of the differences between programming in MicroC/OS-II and a typical desktop operating systems such as Windows or Linux (list at least two differences).
    - How much time you spend in this project?
    - What happen when we change the priority of *malloc_task1*? How does it affect the design of your *malloc* and *free* routines?
    - What is the level of difficulty of this assignment (0 = very easy, 10 = impossible to do)?