Memory Management

Using generic linked list to keep track of free memory blocks

global variables: gheap: list to hold free memory blocks, MEM\_BLOCK\_SIZE: 128 bytes, NUM\_MEM\_BLOCKS: 30

k\_processor.c documentation:

Firstly, this file contains the following global variables:

PCB \*\*gp\_pcbs -- An array of pcb pointers. This array contains all the PCBs for our OS.

PCB \*gp\_current\_process -- A pointer that always points to the current running process.

Afterwards, we declare two queues, the ready queue and the blocked queue. They are both global variables. Then we have our process initialization tables, which contain initialization information for all the processes of our OS.

The function infinite\_loop(void) simply calls release\_processor() forever, and the null process is assigned this procedure. So in essence, all the null process does is call the release\_processor().

Now the function process\_init() initializes all the processes in a system. It calls the set\_test\_proc() function, which fills out the initialization information for all the user processes into a table called g\_test\_procs[], and then process initialization information is copied from there into g\_proc\_table[]. The NULL process is also initialized explicitly beforehand.

Now, we loop through the g\_proc\_table[], and initialize the contents of our gp\_pcbs i.e initializing all the PCBs in our OS. We copy over the pid, priority, and sp, and we initialize all the states to NEW. Afterwards, we push every pcb onto the ready queue.

Now the scheduler(void) function simply picks the pcb of the next process to run. It makes sure that there is a process available on the ready queue to run, and that it is not blocked. Otherwise, it will return the NULL process.

The process\_switch() function simply takes in the previous running process, and sets the gp\_current\_process to running. It takes care of various state information (such as setting gp\_current\_process state to ready, etc.).

The k\_release\_processor(void) simply calls the scheduler to determine the next ready process, and then pushes the old process back to the end of the ready queue.

The method k\_enqueue\_blocked\_on\_resource\_process(PCB \*pcb) sets the state of the pcb passed in to BLOCKED\_ON\_RESOURCE and enqueues it on to the back of the blocked queue.

The method k\_dequeue\_blocked\_on\_resource\_process(void) dequeues the next available process in blocked\_on\_resource queue and returns it.

The method get and set priorites do exactly what is expected: get and set priorities of the required PCBs.

The method check\_preemption() checks if the PCB in the front of the ready queue should preempt the current running process.