Implement an Early Deadline First (EDF) process enqueueing algorithm

Modify the enqueue function on /usr/src/kernel/proc.c so that it enqueues processes in ascending order based on the deadline parameter.

A deadline value of 0 means that the process does not have a deadline and thus needs to be enqueued at the end of correct queue based on its priority.

To aid with debugging enclose the existing enqueue code:

```
/*===========================================================================*
 *                                enqueue                                    *
 *===========================================================================*/
void enqueue(
  register struct proc *rp        /* this process is now runnable */
)
{
...
  rdy_head = get_cpu_var(rp->p_cpu, run_q_head);
  rdy_tail = get_cpu_var(rp->p_cpu, run_q_tail);

  /* Now add the process to the queue. */
  if (!rdy_head[q]) {             /* add to empty queue */
     rdy_head[q] = rdy_tail[q] = rp;              /* create a new queue */
     rp->p_nextready = NULL;          /* mark new end */
  }
  else {                                  /* add to tail of queue */
     rdy_tail[q]->p_nextready = rp;               /* chain tail of queue */
     rdy_tail[q] = rp;                            /* set new queue tail */
     rp->p_nextready = NULL;          /* mark new end */
  }
...
```

So that it checks if the deadline value is other than 0:

```
if(rp->deadline == 0){
 /* Now add the process to the queue. */
          if (!rdy_head[q]) {              /* add to empty queue */
             rdy_head[q] = rdy_tail[q] = rp;              /* create a new queue */
             rp->p_nextready = NULL;           /* mark new end */
          }
          else {                                   /* add to tail of queue */
             rdy_tail[q]->p_nextready = rp;              /* chain tail of queue */
             rdy_tail[q] = rp;                           /* set new queue tail */
             rp->p_nextready = NULL;           /* mark new end */
          }
}else{
        //implement your code here based on the code above
}
```

Otherwise, if you have any mistake, the system will not boot. Be sure to create a VM snapshot before compiling.

Read the code and try to understand the enqueue function, your EDF implementation must manage insertion edge cases and implement an insertion sort loop.

After modifying the proc.c compile the kernel and include it in the OS boot image.

In /usr/src/releasetools compile a new kernel image:

```
# make hdboot
```
Sync & Shutdown Minix and start it to run the new version of the OS.

Use the tests found on the moodle page to test your enqueue function.

The test results must show processes with smaller deadline values executing before those with larger deadline values, in ascending order.

Run with:

```
# ./test_deadline_a
# ./test_deadline_b
```
Note: Press enter after the test finishes to return to the shell.

Test A tests deadlines from 70-100

```
# ./test_deadline_a
mycalllib.h 100
mycalllib.h 90
misc.c 100
sys_edf.c 100
do_setedf.c 100
mycalllib.h 80
misc.c 90
sys_edf.c 90
do_setedf.c 90
mycalllib.h 70
misc.c 80
sys_edf.c 80
do_setedf.c 80
misc.c 70
sys_edf.c 70
do_setedf.c 70
# This is child with deadline 70
This is child with deadline 80
This is child with deadline 90
This is child with deadline 100
_
```

Test B tests deadlines from 30-90 **randomly**

```
# ./test_deadline_b
mycalllib.h 30
misc.c 30
sys_edf.c 30
do_setedf.c 30
mycalllib.h 70
misc.c 70
sys_edf.c 70
do_setedf.c 70
mycalllib.h 80
misc.c 80
sys_edf.c 80
do_setedf.c 80
mycalllib.h 40
misc.c 40
sys_edf.c 40
do_setedf.c 40
# This is child with deadline 30
This is child with deadline 40
This is child with deadline 70
This is child with deadline 80
_
```

Paste screenshots of both test running successfully (ascending order)