Round-Robin con *quando* variable

En esta práctica debéis de implementar el algoritmo **Round Robin** con *quanto* variable. En el proyecto adjunto podéis encontrar un código incompleto, del que únicamente habrá que modificar el fichero `sched_rr_dq.c` .

Distinguiremos entre dos tipo de tareas, las que necesitan usar la CPU durante más de un *cuanto*, o *CPU_BOUND*, y las que no agotan el *quanto* o *INTERACTIVE*. Para ello, en vez de tener una lista de tareas por CPU como en el caso de **RR**:

```
/* Descriptor for a per-CPU run queue */
typedef struct{
    slist_t tasks;      /* runnable task queue */
    task_t* cur_task;   /* Pointer to the task in the CPU */
    task_t idle_task;   /* This CPU's idle task */
    ...
}runqueue_t;
```

tendremos dos listas:

```
/* Descriptor for a per-CPU run queue */
typedef struct{
    slist_t int_tasks;      /* interactive task queue */
    slist_t cpu_tasks;      /* cpu bound task queue */
    task_t* cur_task;       /* Pointer to the task in the CPU */
    task_t idle_task;       /* This CPU's idle task */
    ...
}runqueue_t;
```

que habrá que gestionar de la siguiente forma:

1. Cuando llegue por primera vez una tarea al sistema y se ejecute la función `task_new_rr_dq()` , la insertaremos en el nivel 0 con el cuanto almacenado en la variable global `rr_quantum` .
2. Cuando se busque la siguiente tarea para ser ejecutada a través de `pick_next_task_rr_dq()` , buscaremos primero en la lista *INTERACTIVE* y, si no hay tareas, en la lista de *CPU_BOUND*.
3. Cuando se solicite una tarea para migración a través de `steal_task_rr_dq()` , haremos lo contrario, buscaremos primero en la lista *CPU_BOUND* y, si no hay tareas, en la lista de *INTERACTIVE*.
4. La política de cambio de tipo será la siguiente (gestionado por `task_tick_rr_dq()` y

`enqueue_task_rr_dq()`):

1. Cuando una tarea agote su *quanto*, y no comience inmediatamente un bloqueo, se considerará de tipo *CPU_BOUND*.
2. Cuando una tarea se bloquee sin haber agotado su *quanto*, se considerará de tipo *INTERACTIVE*.
3. En caso de agotar su *quanto* y además coincida con el fin de ráfaga, no cambiara de tipo.
   Para especifical el tipo de una tarea, se ha añadido una nueva variable, `task_type` , a la estructura `task_t` :

```
/* Task descriptor */
typedef struct{
   int task_id;               /* Internal ID for the task*/
   ...
   int runnable_ticks_left;   /* Number of ticks the application...*/
   int task_type;             /* Task type, CPU_BOUND or INTERACTIVE *
/
   int remaining_ticks_slice; /* For the RR scheduler */
   ...
}task_t;
```

Esta variable se actualiza cuando una tarea abandone la CPU, evento que se detecta en la función `task_tick_rr_dq()` , y se utiliza cuando es insertada en la `runqueue` , implementado en la función `enqueue_task_rr_dq()` .

Se recomienda basar la solución en el código original de la práctica (ver solución *RR* con barreras **POSIX**) y comenzar la codificación sin cambio de tipo en las tareas, de manera que la solución coincida con **RR** original. Una vez que estamos razonablemente seguros de la solución con una CPU, se incluirá la gestión del tipo de tarea por parte de `task_tick_rr_dq()` y después las pruebas con 2 CPUs.

En el directorio `examples` podéis encontrar el fichero `exampleDQ.txt` y las soluciones de ejecutarlo con una CPU y *quanto* 1 en el fichero `n-1_q-1.eps` :

```
$ ./schedsim -i ./examples/exampleDQ.txt -q 1 -n 1
```

y con 2 CPUs en los ficheros `n-2-0_q-1.eps` (CPU 0) y `n-2-1_q-1.eps` (CPU 1):

```
$ ./schedsim -i ./examples/exampleDQ.txt -q 1 -n 2
```