Round-Robin Multitasking

RTX Kernel can be configured to use Round-Robin Multitasking (or task switching). Round-Robin allows quasi-parallel execution of several tasks. Tasks are not really executed concurrently but are **time-sliced** (the available CPU time is divided into time slices and RTX Kernel assigns a time slice to each task). Since the time slice is short (only a few milliseconds) it appears as though tasks execute simultaneously.

Tasks execute for the duration of their time-slice (unless the task's time slice is given up). Then, RTX Kernel switches to the next task that is **ready** to run and has the **same priority**. If no other task with the same priority is ready to run, the currently running task resumes it's execution. The duration of a time slice may be defined by the **RTX Config.c** configuration file.

The following example shows a simple RTX program that uses Round-Robin Multitasking. The two tasks in this program are counter loops. RTX Kernel starts executing task 1 which is the function named **job1**. This function creates another task called **job2**. After **job1** executes for its time slice, RTX Kernel switches to **job2**. After **job2** executes for its time slice, RTX Kernel switches back to **job1**. This process is repeated indefinitely.

```
#include <RTL.h>

int counter1;
int counter2;

void job1 (void) __task;
void job2 (void) __task;

void job1 (void) __task {
   os_tsk_create (job2, 0);  /* Create task 2 and mark it as ready */
   while (1) {               /* loop forever */
      counter1++;            /* update the counter */
  }
}

void job2 (void) __task {
  while (1) {                /* loop forever */
    counter2++;              /* update the counter */
  }
}

void main (void) {
   os_sys_init (job1);       /* Initialize RTX Kernel and start task 1 */
   for (;;);
}
```

Note

- Rather than wait for a task's time slice to expire, you may use one of the **system wait** functions or the **os_tsk_pass** function to signal RTX Kernel that it can switch to another task. The system wait function suspends the current task (changes it to the **WAIT_xxx** State) until the specified event occurs (and the task is changed to the **READY** State). During this time, any number of other tasks may run.