TASK - Cooperative Multi-threading Function Library for Microsoft C

This package implements a very fast, cooperative, non-preemptive multi-threading library for programs written in Microsoft C. Multi-threaded programs can run multiple tasks in the form of C callable functions. Tasks are run in a round-robin fashion. Each task must, at some point, either give up control or return. When a task returns, it can no longer execute. One task may suspend or resume another task. A special function is provided for aborting all task.

This general purpose library allows you, for example, to have one task that waits for keyboard input while having other tasks perform other functions (i.e., graphics, etc.) All tasks have their own stack and run in the same address space so they may share global variables and structures.

SYNOPSIS

```
   #include "task.h"                // TASK package definitions

   short task_spawn(TASK *, ...);         // A NULL terminated list of C
functions
   void task_switch(void);                // Called by tasks to give up CPU
   short task_suspend(TASK *);            // Suspend task on next task switch
   short task_resume(TASK *);       // Resume task after next task switch
   void task_abortall(void);        // Abort all tasks immediately
   char *task_error(short n);       // Error message
```

DESCRIPTION

This package implements non-preemptive multiple threading (coroutines) in C. A set of C functions, called tasks, is defined and control is passed to them by task_spawn. The tasks cooperatively relinquish the CPU by periodically making calls to task_switch.

YOU MUST RESERVE ENOUGH SPACE ON THE PROGRAM STACK FOR EACH TASK + MAIN. Typically 2K per task is enough to do most things unless you use a lot of dynamic (stack) arrays - See STACK_SIZE parameter for LINK.EXE). If one of the tasks consumes more than its share of stack, your program may crash.

Before calling task_spawn, you can set a global USHORT task_stack_size to adjust the size of each task stack (the default is 2000 bytes). For example, if you have 4 tasks, your program should have at least 10,000 bytes of stack (the main program + 4 tasks).

task_spawn - Takes a NULL terminated list of C functions which constitute the complete set of functions that will participate as tasks. Each task has its own stack and is treated as one thread. Control passes round robin to each task. If a task returns, it is removed from the active list of tasks. If a task calls task_switch, it temporarily gives up control of the CPU and the next task is run. After a task calls task_switch, the other tasks will run. When task_switch returns, the task may continue processing. There are no more tasks when all tasks return. When this happens, the task_spawn function finally returns. Thus, the main program (the one that calls task_spawn) is "suspended" at the task_spawn call until all tasks return.

task_switch - Each task calls this to give up the CPU for a while. When task_switch returns, the task may continue executing. Eventually every task function must return. Inside a task, you may call this at any point and any number of times.

task_suspend - Suspend a task. Argument is the function specified in task_spawn. If argument is NULL, the current task is suspended. Once suspended, the task is not given the CPU until task_resume is called.

task_resume - Resumes a task. Argument is a function specified in task_spawn. If the argument is NULL or the current task, no action is taken and control returns immediately to the caller.

task_error - Returns an error message string given a TSK_xxx error code. The TSK_xxx error code is a code returned by one of the tasking functions.

task_abortall - May be called by any task. This aborts execution of all tasks and returns control to the main program (i.e., task_spawn returns to the function that called task_spawn).

RETURN CODES

See TASK.H for a list of return codes (TSK_xxx).

task_switch returns control after all other tasks have had their chance to run. task_spawn return TSK_COMPLETE if all tasks successfully exited, or a negative number if an error occurred. Use task_error to get error message string.

SEE ALSO

Microsoft C 6.0 Release notes, README.DOC, Part 3.
Microsoft C Runtime Library Reference, 1990.

COMPILING AND LINKING

Compile and link with the large memory model. Include TASK.H in your C source code and link with the RDXTASK.LIB library.

Use the following pragma with the Microsoft C compiler.

```
#pragma    optimize("elg", off)
```

This library is designed for the Microsoft C 6.0 compiler.