# A Cooperative Executive (ACX)

## Introduction

ACX is a simple, non-preemptive, multi-threading kernel that supports up to 8 threads. When created, each thread is assigned an ID (between 0 and 7 inclusive), its own stack area (associated with its ID), and an entry point. ACX uses a single timer and its corresponding interrupt vector to support thread delays (measured in “system tick” units), and an absolute tick counter that is simply increments each system tick, wrapping on overflow. Stack sizes are defined statically in the file **acx.h** while source code is in the files **acx.c** and **acx\_asm.S** (written in assembly language).

## Kernel Definitions

Some of the types, definitions and macros declared in acx.h:

#define x\_getTID() (x\_thread\_id) //x\_thread\_id is a global declared in acx.c

#define MAX\_THREADS 8

#define NUM\_THREADS 8

typedef uint8\_t byte;

typedef void (\*PTHREAD)(void);

typedef struct {

uint8\_t \*sp;

uint8\_t \*pStackBase;

} STACKCONTROL;

## Kernel Functions

The following functions are supported by ACX, and are callable from C functions.

|  |  |
| --- | --- |
| x\_init | x\_init()  Initializes kernel data structures, sets stack pointer to Thread0 stack area returns to caller. The calling function becomes Thread0. |
| x\_new | void x\_new(byte tid, PTHREAD pthread, Byte isEnabled)  Assigns a function pointer (pthread) to a thread ID and sets its initial enable status. If the thread ID is the same as the calling thread, then the calling thread is replaced and rescheduling takes place.  Note: This function **MAY NOT** be called from an interrupt handler. |
| x\_yield | void x\_yield()  Invokes kernel scheduler to give other threads a chance to run. Does not change state of current thread. If no other threads are READY, then current thread (the calling thread) is restored to execution.  Note: This function **MAY NOT** be called from an interrupt handler. |
| x\_delay | void x\_delay(int ticks)  Delays the calling task by the specified number of system “ticks”. Actual delay depends on tick frequency, set at system initialization, and the latency of other threads scheduled before the calling thread when it becomes READY.  Note: This function **MAY NOT** be called from an interrupt handler. |
| x\_suspend | void x\_suspend(int tid)  Suspends the specified thread by setting its suspend status bit.  Note: This function **MAY** be called from an interrupt handler. |
| x\_resume | void x\_resume(int tid)  Resumes specified thread by clearing its suspend status bit.  Note: This function **MAY** be called from an interrupt handler. |
| x\_disable | void x\_disable(int tid)  Disables specified thread by setting its disable status bit. This makes the puts the thread into a BLOCKED state. For now this bit behaves just like the suspend status, only providing another level of “suspend”.  Note: This function **MAY** be called from an interrupt handler. |
| x\_enable | void x\_enable(int tid)  Enables specified thread by clearing its disable status bit.  Note: This function **MAY** be called from an interrupt handler. |
| x\_gtime | long gtime()  Returns the 32-bit time as recorded on the absolute tick counter. |

## ACX Data

ACX reserves data structures to keep track of each thread’s state. The following memory areas are used (the names are not necessarily those used in the code):

|  |  |
| --- | --- |
| Stack area | One stack area for each thread. Sizes are defined statically in acx.h |
| Array of STACKCONTROL structures | A table of structures, each with two members: a constant pointer to the base of a stack, and a pointer to the current top of stack. |
| Delay values | A table of 16-bit delay values that record current delay of each thread |
| Disable Status | A byte that stores the disable status (1=disabled), 1 bit per thread. Bit 0 corresponds to Thread 0. |
| Suspend Status | A byte that stores the suspend status (1=suspended), 1 bit per thread. Bit 0 corresponds to Thread 0. |
| Delay Status | A byte that stores the delay status (1=delayed), 1 bit per thread. Bit 0 corresponds to Thread 0. |
| Absolute Tick Counter | A 32-bit value that stores the current count of system ticks. Non-decreasing, initialized at kernel initialization and counts up once per system tick interrupt. |



