Clarion

Cooperative Threading
in a
Preemptive Environment

SoftVelocity

www.softvelocity.com
(800) 270-4562

One of the advantages of Clarion 6 is that it is still possible to create an application that emulates the behavior associated with cooperative threading.   Clarion 6 provides the necessary hooks so that you can decide whether your threads are preemptive, cooperative, or a mixture of the two. This article introduces the concept and provides code samples.

Cooperative Threading Extension templates

All applications created prior to Clarion 6 had all threads other than the first one work in a cooperative manner. All new applications, and applications migrated to Clarion 6 will, by default, generate fully preemptive thread based applications.

If you decide to emulate the cooperative thread system, you will still have to convert some code affected by the change in the underlying thread and memory management model (see the *Multi-Threaded Programming* PDF).

If you use the Global Cooperative Threading extension template you are not limited to only have cooperative threads. You can have some or all threads preemptive. To make a procedure thread preemptive you need to activate the “Preemptive Procedure” procedure extension.

If your application is generated with preemptive threads, you can still make individual threads co-operative by adding the Global Cooperative Threading extension and turning off the cooperative option. Then, on each “Preemptive Procedure” procedure extension, turn off the preemptive feature to make the target procedure thread cooperative.

Cooperative Threads using Hand Code

To allow you to create cooperative threads under Clarion you need to add the following code to your program

```
PROGRAM
  INCLUDE('CWSYNCHM.INC'),ONCE
  INCLUDE('CWSYNCHC.INC'),ONCE
  MAP
    UnlockProc()
    LockProc()
    LockedProc(),BYTE
  END

ThreadLocker Mutex

CooperationClass CLASS,THREAD
Preemptive          BYTE,PRIVATE
Locked              BYTE,PRIVATE

PreemptiveThread    PROCEDURE(BYTE newState)
IsPreemptive        PROCEDURE(),BYTE
IsLocked            PROCEDURE(),BYTE
Wait                PROCEDURE
Release             PROCEDURE
                 END

  CODE
    SYSTEM{PROP:UnlockThreadHook} = ADDRESS(UnlockProc)
    SYSTEM{PROP:LockThreadHook} = ADDRESS(LockProc)
    SYSTEM{PROP:ThreadLockedHook} = ADDRESS(LockedProc)
    ! Do everything

CooperationClass.PreemptiveThread PROCEDURE(BYTE newState)
  CODE
    IF newState AND NOT SELF.Preemptive
      ThreadLocker.Release()
    ELSIF NOT newState AND SELF.Preemptive
      SELF.Preemptive = newState
      ThreadLocker.Wait()
    END
    SELF.Preemptive = newState

CooperationClass.Wait PROCEDURE()
  CODE
    IF NOT SELF.Preemptive
      ThreadLocker.Wait()
      SELF.Locked = TRUE
    END

CooperationClass.Release PROCEDURE()
  CODE
    IF NOT SELF.Preemptive
      SELF.Locked = FALSE
      ThreadLocker.Release()
    END
```

```
CooperationClass.IsPreemptive PROCEDURE()
  CODE
    RETURN SELF.Preemptive

CooperationClass.IsLocked PROCEDURE()
  CODE
    RETURN CHOOSE(SELF.Preemptive,FALSE,SELF.Locked)

UnlockProc PROCEDURE
  CODE
    CooperationClass.Release()

LockProc PROCEDURE
  CODE
    CooperationClass.Wait()

LockedProc PROCEDURE
  CODE
    RETURN CooperationClass.IsLocked()
```

If you want a thread to be preemptive, then you need to add the following line of code near the start of the procedure that is called when the thread starts

```
CooperationClass.PreemptiveThread(TRUE)
```

Functions that Unlock a Thread

You need to be aware that these functions will call UNLOCKTHREAD on entry to them and LOCKTHREAD on exit. You cannot assume that static variables do not change their contents across calls to these functions.

ACCEPT
COLORDIALOG
DELAY
FILEDIALOG
FONTDIALOG
MESSAGE
OPEN(File) for ODBC driver
RUN when waiting for process to terminate
YIELD

Warning: The Construct method of a threaded class is called before a thread is locked and the Destruct method after the thread is unlocked.

Note: In an MDI based application you must not lock the main thread