Using external C code with pre-emptive multi-tasking in Xojo

Thomas Tempelmann

MBS Xojo Conference, Munich

September 7, 2018

Challenges

- Xojo cannot run code in multiple threads concurrently, due to lack of securing the lowest data structures (Objects, includings Strings and Arrays) accordingly.
- Any parallel processing (on multiple CPU Cores) must be performed outside Xojo code, e.g. in a library written in C.

Binding with the C code

- On macOS, create a dylib; on Windows a DLL
- The lib exports static C functions
- Add the lib file to the built app using Build Automation
- Link to the functions from Xojo with declares

Threads and Tasks

- The C code sets up a number of pthreads (POSIX Threads)
- Xojo creates Tasks, which are described in Structures. These get passed to the C code, which adds them to a queue (list).
- Each running Thread waits for Tasks to arrive, and processes one of them at a time.
- Once a task has been handled by a thread, it sets a flag, which the Xojo code polls via a Timer or in a Xojo Thread.

Task data structure

- Example: Pass data to the C code and have it count the bits of every byte.
- Parameter exchange:
  - From Xojo to C lib:
    - Address of data
    - Length of data
  - From C lib to Xojo:
    - Result (number of counted bits)

C code skeleton

```
typedef struct sharedTask {
   uint16_t ident;         // must equal 0xB119
   uint16_t ownSize;       // must equal sizeof(sharedTask)
   privateData internal;
   uint8_t status;         // 0: ready, 1: working, 2: finished
   // ... add your own data here
   char *inputBuf;
   uint32_t inputLen;
   uint64_t result;
} sharedTask;

typedef struct {  // links the sharedTask records into a queue
   uint16_t ident;         // must equal 0xB117
   uint16_t ownSize;       // must equal sizeof(privateData)
   struct sharedTask *next;
} privateData;
```

C code skeleton

```
DLL extern boolean helper_createThreads (int threadCountIn)
{
   gStop = 0;
   gThreads = calloc (threadCountIn, sizeof(gThreads[0]));
   gThreadCount = threadCountIn;

   for (int n = 0; n < gThreadCount; ++n) {
      pthread_create (&gThreads[n], NULL, threadRunner, n);
   }

   return true;
}
```

C code skeleton

```
DLL extern boolean helper_enqueue (sharedTask *task) {
   task->internal.next = NULL;

   // Add the task to the queue.
   pthread_mutex_lock(&gInputLock);
   if (!gNextTask) {
      gNextTask = task;
   } else {
      gLastTask->internal.next = task;
   }
   gLastTask = task;
   gPendingTaskCount += 1;
   pthread_mutex_unlock(&gInputLock);

   // Signal the waiting threads so that one picks up the task
   pthread_cond_signal (&gInputCond);

   return true;
}
```

C code skeleton

```
static void *threadRunner(void *threadNum) {
   while (!gStop) {
      sharedTask *myTask;

      pthread_mutex_lock (&gInputLock);

      // Wait for a new task (added via helper_enqueue)
      while (gNextTask == NULL) {
         pthread_cond_wait (&gInputCond, &gInputLock);
      }

      // Pick next task from queue
      myTask = gNextTask;
      gNextTask = myTask->internal.next;
      gPendingTaskCount -= 1;

      pthread_mutex_unlock (&gInputLock);
```

C code skeleton

```
    …

    myTask->status = 1;
    gBusyTaskCount += 1;

    boolean ok = false;

    //
    // Do the task's work here, using the information in myTask
    //

    ok = true;

    // Mark the task finished
    myTask->status = (ok) ? 2 : 3;
    gBusyTaskCount -= 1;
}
```

Xojo code skeleton

SharedTask

	Declaration	Offset	Size
⊖	ident as UInt16	0	2
⊖	ownSize as UInt16	2	2
⊖	internal as InternalData	4	8
⊖	status as UInt8	12	1
⊖	filler as String * 3	13	3
⊖	inputBuf as Ptr	16	4
⊖	inputLen as UInt32	20	4
⊖	result as UInt64	24	8
⊕			32

Xojo declares

HelpersLib As String

Platform	Language	Value
Windows	Default	C_Helper.dll
OS X	Default	@executable_path/../Frameworks/C_Helper.dylib

```
declare function f_ lib HelpersLib alias "helper_initialize" () as Boolean

if not f_ () then
   // something went wrong
   break
end if
```

Class ProcessingTask

```
Constructor(input as MemoryBlock, userInfo as Variant)
    mInput = input // let's keep the MemoryBlock around until we're finished with this task

    mUserInfo = userInfo

    // set up internal data
    mTaskData.ident = &hB119
    mTaskData.ownSize = mTaskData.Size
    mTaskData.internal.ident = &hB117
    mTaskData.internal.ownSize = mTaskData.internal.Size

    // set up our user data
    mTaskData.inputBuf = mInput
    mTaskData.inputLen = mInput.Size
    mTaskData.result = 0
```

```
Start
    mTaskData.status = 0
    C_Helper.Enqueue mTaskData
```

Using ProcessingTask

```
call C_Helper.CreateThreads (4)

dim testData as String = TestData

// set up each task and run a thread on it
for taskNum as Integer = 0 to 15
  mTasks.Append = new ProcessingTask (testData, taskNum)
next

// now start each task, which in turn passes its data to the helper's threads
for taskNum as Integer = 0 to mTasks.Ubound
  mTasks(taskNum).Start
next

pollTimer.Mode = Timer.ModeMultiple
```

Polling Task results

```
if mTasks.Ubound >= 0 then
  for i as Integer = 0 to mTasks.Ubound
    dim task as ProcessingTask = mTasks(i)
    if task.Status = 2 then
      // finished - we could add new Tasks to mTasks now
    end if
  next
end if
```

References

- Sample Code:
  - http://files.tempel.org/RB/Threading

- Forum posts about handling pre-emptive threads
  - https://forum.xojo.com/49704
  - https://forum.xojo.com/20313