```
outw[0]= {-0.23686,0.848603,0.871922,0.0992451,-0.856209}
   task=   sin  results=
outw[0]= {0.999734,0.531563,-0.172071,-0.931296,-0.750942}
   task=   cos  results=
outw[0]= {-0.166893,0.767707,0.925874,0.114511,-0.662994}
   worker closing down due to group release
   worker closing down due to group release
```

Parallel Processing With Large Datasets

In the preceding section we synthesized the input data. In the real-world, your input data would most-likely be in an existing wave and you would have to copy it to a data folder to put into the input queue.

If your input data is very large, for example, a 3D stack of images, copying would require too much memory. In that case, a good choice is to pass the input directly to the thread using parameters to the thread worker function and use the output queue to return output to the main thread.

To do this you can use the **Parallel Processing - Thread-at-a-Time Method** and the output queue to return results.

Preemptive Background Task

In this example, we create a single worker thread that runs while the user does other things. A normal cooperative background task retrieves results from the preemptive thread. Although the background task will sometimes be blocked (as described in **Background Tasks** on page IV-319) the preemptive worker thread will always be running or waiting for data.

Another example of this kind of multitasking can be found in the "Slow Data Acq" demo experiment referenced under **More Multitasking Examples** on page IV-339.

In some cases, it may be possible to run two instances of Igor instead of using a preemptive background task. Running two instances is far simpler, so use that approach if it is feasible.

We put the code for the background tasks in an independent module (see **The IndependentModule Pragma** on page IV-55) so that the user can recompile procedures, which is done automatically when a recreation macro is created, without stopping the background task.

You might use a preemptive background task is when you have lengthy computations but want to continue to do other things, such as creating graphics for publication. Although you can do anything you want while the task runs in the experiment, if you load a different experiment, the thread is killed.

For this example, our "lengthy computation" is simply creating a wave of sine values which is not lengthy at all and consequently there is no reason for using a preemptive thread in this case. To simulate a lengthy computation, the code delays for a few seconds before posting its results.

The named background task checks the output queue every 10 ticks, when it is not blocked, and updates a graph with data retrieved from the queue.

Independent modules can not be defined in the built-in procedure window so paste the following code in a new procedure window:

```
#pragma IndependentModule= PreemptiveExample

ThreadSafe Function MyWorkerFunc()
   do
      DFREF dfr = ThreadGroupGetDFR(0,inf)
      if( DataFolderRefStatus(dfr) == 0 )
         return -1              // Thread is being killed
      endif

      WAVE frequencies = dfr:frequencies   // Array of frequencies to calculate
      Variable i, n= numpnts(frequencies)

      for(i=0;i<n;i+=1)
         NewDataFolder/S resultsDF
         Make jack= sin(frequencies[i]*x)
```

```
          Variable t0= ticks
          do
             // waste cpu for a few seconds
          while(ticks < (t0+120))

          // ThreadGroupPutDF requires that no waves in the data folder be referenced
          WAVEClear jack

          ThreadGroupPutDF 0,:             // Send current data folder to input queue
       endfor

       KillDataFolder dfr      // We are done with the input data folder
    while(1)

    return 0
End

Function DisplayResults(s)    // Called from cooperative background task
    STRUCT WMBackgroundStruct &s

    DFREF dfSav= GetDataFolderDFR()

    SetDataFolder root:testdf
    NVAR threadGroupID
    DFREF dfr = ThreadGroupGetDFR(threadGroupID,0) // Get free data folder from queue
    if( DataFolderRefStatus(dfr) != 0 )
       // Make free data folder a regular data folder in root:testdf
       MoveDataFolder dfr, :

    // Give data folder a unique name
       String dfName = UniqueName("Results", 11, 0)
       RenameDataFolder dfr, $dfName

       WAVE jack = dfr:jack       // This is the output from the thread
       AppendToGraph/W=ThreadResultsGraph jack
    endif

    SetDataFolder dfSav
    return 0
End
```

And put this in the main procedure window:

```
Function DemoPreemptiveBackgroundTask()
    DFREF dfSav= GetDataFolderDFR()

    NewDataFolder/O/S root:testdf // thread group ID and result datafolders go here

    Variable/G threadGroupID= ThreadGroupCreate(1)

    ThreadStart threadGroupID,0,PreemptiveExample#MyWorkerFunc()

    // MyWorkerFunc is now running and waiting for input data
    // now, let's give it something to do
    NewDataFolder/S tasks
    Make/N=10 frequencies= 1/(10+p/2+enoise(0.2))// array of frequencies to calculate
    WAVEClear frequencies

    ThreadGroupPutDF threadGroupID,:        // thread is now crunching away

    // Results will be appended to this graph
    Display /N=ThreadResultsGraph as "Thread Results"

    // ...by this named task
    CtrlNamedBackground ThreadResultsTask,period=10,proc=PreemptiveExample#DisplayResults,start

    SetDataFolder dfSav    // restore current df
End

Function PostMoreFreqs()
    NVAR threadGroupID = root:testdf:threadGroupID

    NewDataFolder/S moretasks
    Make/N=50 frequencies= 1/(15+p/2+enoise(0.2))  // array of frequencies to calculate
    WAVEClear frequencies

    ThreadGroupPutDF threadGroupID,:               // thread continues crunching
End
```