Subject: Re: Multitasking??
Posted by p.sommer on Tue, 23 Nov 2004 17:11:14 GMT
View Forum Message <> Reply to Message

JD Smith <jdsmith@as.arizona.edu> wrote in message
news:<pan.2004.11.22.16.55.49.864974@as.arizona.edu>...
> On Sun, 21 Nov 2004 08:53:48 -0800, Paul wrote:
>
>> Ken Mankoff <mankoff@yahoo.com> wrote in message
news:<Pine.OSX.4.61.0411191245570.26987@gouda.local>...
>>> On Fri, 19 Nov 2004, Henry Roe wrote:
>>>> Just an idle under-caffeinated thought: Is there a simple way for
>>>> two IDL processes to communicate? (besides writing to a disk on
>>>> file) If so, then the two GUI's could actually be separate IDL
>>>> processes.
>>>
>>> You can use the SOCKET command for communication w/o files.
>>>
>>> -k.
>> In addition, you can use IDL's shared memory (shmmap) for this. I
>> have done this for camera control where one process presents a GUI and
>> the 2nd process handles the setting of a buffer, acquiring data and
>> rapid display. This way, you don't loose focus on your main GUI. I
>> suspect your subGUI could be running in process two in a similar way
>> my 'slave' process was running (no GUI).
>>
>> Basically, what you do is set up a structure variable that contains
>> data fields that need to be shared between processes. I then created
>> a main procedure that simply spawned mainGUI to run under IDL's
>> runtime engine (idlrt). From mainGUI, I had a button that started the
>> data acquistion. In this button's event handler, I spawned off the
>> 2nd process also under the idlrt engine. At this point you now have
>> three processes going - main, mainGUI and in your case, subGUI.
>>
>> There is a very simple example on RSI's user contrib site that I put
>> together to prototype the final solution described. See:
>> http://www.rsinc.com/codebank/search.asp?FID=259
>>
>> If you choose to try the above and end up having questions, do not
>> hesitate to contact us. Thanks and good luck.
>
> Is there an advantage to using a second IDL process for the data
> acquisition? I would think that since you've probably had to write
> the camera (or whatever I/O) interface code in C, you might as well
> have a data capture tool that runs stand-alone, and can communicate
> with an IDL GUI process via shared memory (e.g. an SHMVAR variable).
> A socket could also be used, but shared memory is probably the fastest
> way to pull any amount of data in.

>
> JD

Hi JD,

The prospect I was working with at the time had a requirement to pull fairly large image data (~2 M-byte) over Ethernet into IDL. The prospect also wanted the IDL mainGUI to handle other tasks beyond grabbing/displaying image frame data. In addition, there was a desire to componentize the architecture.

So, what I ended up doing was building an acquisition component (object) that implemented a SOCKET protocol (also object). This protocol could communicate with the camera. It pinged the camera for new data, set a buffer in shared memory, etc, etc. Then, the mainGUI (an iTool) would occasionally check a status byte (via Widget Timer event) to see if new data was available. The user also had a button to query the status byte.

The end result was pretty neat...almost like a threaded application where GUI interactivity was retained even during the heavy SOCKET activity that was occurring in the 'background' (process 2). As you can imagine, displaying the data once in the shared buffer was very fast as well -- yes, even using the iTool framework. Seriously, I was amazed how well the underlying iTool system performed here. All I needed to do was call the tool's internal setData method and like magic, I got a refreshed display that would automatically scale, provide access to the iTool's image property widget, etc.

Admittedly, I probably took the long way around to solve this problem but is was interesting mixing some of IDL's more recent features to address this challenge. While I can't officially share all that code, I could make the iTool stand alone if you guys are interested. At least it's a real world example on how one might create a custom iTool. Bottom line is that It's not as challenging as one might think and they really do lend themselves well for rapid application development (for the right problem).

Best regards,
Paul