Dont PtLeave me Hanging!

Authored by: **John Fehr**
Updated by: **Mikhail Nefedov**

Many GUI applications are responsive: you click on a widget, the application responds immediately.

Other applications are much less responsive. The application may have to perform time-intensive operations or communicate with remote services which may block the application for seconds at a time. One way to make this type of application more responsive is to use multiple threads. With at least one thread always available to handle user interaction, the other threads are performing the blocking or lengthy operations.

But now two threads could try to access or update the same widget at the same time. If this were to happen, the application would segment violate. This article deals with how to solve this problem.

As mentioned above, we might have a program that calls a function that does some time-intensive work. It then modifies one of its widgets, most likely based on the outcome of the work just done.

We'll use a simple example to clarify our points, using a "**for**" loop as the time-intensive part, and a **PtWindow** title change as the widget modification. In a real application, of course, something more useful would be done.

(You can compile/link these examples with 'qcc -o testapp testapp.c -lph')

```
#include <stdio.h>
#include <errno.h>
#include <time.h>
#include <photon/PtWindow.h>
void *thread_func(void*);
void my_widget_func(PtWidget_t *w,char *text);
int main(int argc,char *argv[])
{
     PtWidget_t *win;
     PtArg_t args[2];
     PhDim_t dim={200,20};
     pthread_t thread;
     PtSetArg(&args[0],Pt_ARG_DIM,&dim,0);
     win = PtAppInit(NULL,NULL,NULL,1,args);
     PtRealizeWidget(win);
     pthread_create(&thread,NULL,thread_func,win);
     PtMainLoop();
}
void my_widget_func(PtWidget_t *w,char *text)
{
     int eval;
     volatile int z;
     for (z=0;z<0x1000000;z++) z=z;
     PtSetResource(w,Pt_ARG_WINDOW_TITLE,text,0);
}
void *thread_func(void *data)
{
```

```
     PtWidget_t *win=(PtWidget_t*)data;
     char title[80];
     while (1)
     {
          time_t ltime=time(NULL);
          strftime(title,80,"Its %H:%M:%S", localtime(&ltime));
          my_widget_func(win,title);
          sleep(5);
     }
}
```

If you compile and run this program, you'll notice that the application dies of a segment violation (**SIGSEGV**) after five seconds. Why? QNX Photon microGUI functions aren't thread safe -- our call to **PtSetResource**() trips something up internally, and KABOOM!

The people at QNX Software have been kind enough to provide an answer: **PtEnter**() locks the Photon library, and **PtLeave**() unlocks the Photon library. We can make any kind of Photon library call between those two function calls. To make our example work, change the **PtSetResource**() call to:

```
if ((eval=PtEnter(Pt_EVENT_PROCESS_PREVENT))>=0)
{
       PtSetResource(w,Pt_ARG_WINDOW_TITLE,text,0);
       PtLeave(eval);
}
```

We're simply calling **PtEnter**(), making sure it's not returning an error, then calling our **PtSetResource**() function, and calling **PtLeave**(). Works like a charm.

Let's complicate things a bit. It could be anything but, for this example, let's say you need to call our **my_widget_func**() function when the widget is exposed. Let's also assume that you can't, for whatever reason, move the **PtEnter**/**PtLeave**() calls outside of your widget function.

Here's the code for our callback:

```
int event_handler(PtWidget_t *widget,void *data,PtCallbackInfo_t
*cbinfo)
{
     my_widget_func(widget,"Time to expose");
     return Pt_CONTINUE;
}
```

and of course we need to add the handler to the widget by inserting the following line after the **PtRealizeWidget**() function call:

```
PtAddEventHandler(win,Ph_EV_EXPOSE,event_handler,NULL);
```

Now, we'd expect our window title to change temporarily to 'Time to expose' every time we dragged another window across it. (The window widget will get an expose event every time part of it is exposed.) But nothing happens!

This is because the Photon library is already locked by the **PtMainLoop**() thread. So when we try to lock it again with **PtEnter**(), it fails. (**PtEnter**() and **PtLeave**() calls are not recursive.)

We can easily solve this problem by checking if the value returned by **PtEnter**() is -**EDEADLK**.

```
if ((eval=PtEnter(Pt_EVENT_PROCESS_PREVENT))>=0 || eval==-EDEADLK)
{
     PtSetResource(w,Pt_ARG_WINDOW_TITLE,text,0);
     if (eval>=0) PtLeave(eval);
}
```

This works because **PtEnter**() is guaranteed to fail with **-EDEADLK** if the calling thread already owns the lock. As a result, it's safe for this thread to modify widgets or Photon function calls.

Also notice that since the lock was not obtained in this function, it is not released here either. In the code snippet above, calling **PtLeave**() when a **PtEnter**() has not been called or has returned -**EDEADLK**, will lead to an eventual segment violation (**SIGSEGV**).

Problem solved! Now everything works as we want it to. :)

Happy safe multi-threaded widget coding!