Subject: Graduation thesis - Camouflage - a replacement of(a part of) Chameleon
Posted by andrei_natanael on Wed, 11 Nov 2009 17:27:12 GMT
View Forum Message <> Reply to Message

I've talked here a bit about my degree thesis. Now it's time to start working on. I will be glad to receive any feedback about implementation, maybe if i get it done very well it may replace a part of upp chameleon which IMHO it's a bit messy, think if you have to start doing chameleon stuff for a new system, from where you start? There's no standard interface you should implement so I'm going to work on an interface which should be implemented on any system, debugging it easy and having a base from where you start when implementing it on a new system, say MacOSX.
Here I'm presenting some ideas of how it should work, pro and cons about some implementations ways. More to come as I think about it...

Name: Camouflage
Language: C/C++
Legend: + pro, - cons, ~ between pro and cons

Pro/Cons:
C Language:
+ may be used easily from other languages
+ runtime linking (dlopen,LoadLibary, etc.)
- poor standard library

C++ language:
+ OOP
~ standard library
~ no runtime linking using standard methods(possible if exporting C interfaces)

Library usage:
1. U++ linked with library at runtime and library is linked statically(because C++) with KDE(QT)
2. U++ static linked with library and library linked dynamically with GTK+, WINAPI, Cocoa*(*is that possible?)
3. U++ static linked with library which is static linked to KDE, GTK+, WINAPI, Cocoa

Explanations for library usage:
1. In case one i have to provide a library for KDE and one for GTK, the program may use gtk or kde lib based on current environment
(this is related to Linux, because other systems doesn't provide multiple Desktop Environments)
+ one program, many interfaces (gtk, kde, x11)
- multiple libraries, it may be a bit complex and the program will depend on these dll(so) (but portability is achieved)
- runtime linkage is not as fast as static linkage(compile time), but it's used on other platforms too to provide portability(i.e. Chameleon in Windows)
How it work: Program will check current environment if it's gtk then load camouflage lib bassed on gtk, if kde then load kde based lib else use only X11 for widgets draw

2. The case two is(should be) the actual U++ state, but it lack KDE style (and Cocoa, but that's another story)

IIRC U++ is static linked with GTK, you may use it without gtk(X11 only) with NOGTK flag.
- will miss KDE interface, because KDE is C++ and we cannot load C++ libs because dlopen doesn't support it and it's not possible because C++ name mangling
+ it will be possible to use gtk interface or X11 only based on program arguments, so you may have a single program for both gtk and X11(only)
How it work: if gtk is available the program with use it else it will use only X11 and will be possible to control gtk or X11 usage based on parameters sent to program
i.e. ./myprogram --X11-view # and upp lib will not load gtk libs

3. - loosing portability because it will depend directly on available libraries
- multiple binaries for program on one platform(gtk, kde, X11)

After having these pro and cons I'm thinking to make it using second approach (U++ way), but that means i will have to drop KDE support or if someone have an idea how to load a static member function of a C++ class(QApplication::style()) then the problem is like solved.
