Subject: Re: Sound in linux (& generic launchers)
Posted by rainbowsally on Thu, 18 Dec 2014 03:23:57 GMT
View Forum Message <> Reply to Message

Yuh know?

The lack of standards in Linux is really a problem.

Here's the problem, linux guys.

ls /usr/share/sounds/freedesktop/stereo/*-warning.*
ls /usr/share/sounds/freedesktop/stereo/*-information.*
ls /usr/share/sounds/freedesktop/stereo/*-question.*

See it? The question sound (a symlink to one of the basic sounds) is inconsistent with the other two dialog sounds and is probably misnamed.

This is the sound string I'd recommend for "question" that should have the greatest probability of being portable.

```
void BeepQuestion()
{
#ifdef PLATFORM_WIN32
 MessageBeep(MB_ICONQUESTION);
#else
 LinuxBeep("warning");
#endif
}
```

There may be a ton of interesting sounds for KDE and possibly GTK in the directory above these, but as far as standards go, freedesktop.org is the closest we've got to a standard at this point.

////////// And Generic Launchers ////////

[There's a question at the bottom.]

There are MANY more problems with linux GUI toolkits, though. And I'm not sure that this is the place to bring them up, but in GUIs we expect things to run concurrently. We don't expect things like playing sounds to block (that is, to wait until the sound is finished) before painting a window. And we don't want to have to create a new library of new non-standard sounds that blow our applications up in size due to the impracticality of compressing these things.

There are ways to accomplish this by way of shell calls, but (again) there is no basic binutils or coreutils level of linux utility to do this.

Examples of applications that do shell calls that run concurrently, by way of reparenting the called

process to 'init' are kshell3, 4, or 5. I have kshell4. It genuinely returns before the sound is finished playing.

In kde compare [Note the lowercase Q switch. Don't use P].
play -q /usr/share/sounds/KDE-Sys-Log-In-Long.ogg
to
kshell4 play -q /usr/share/sounds/KDE-Sys-Log-In-Long.ogg
and if you try the launch utility, try that as well.
launch play -q /usr/share/sounds/KDE-Sys-Log-In-Long.ogg

And for superuser, there kdesu, kdesudo, gksu, and xdg-su, and others which also reparent the called application to init. But they are meant for running GUIs from the commandline as superuser. That's overkill and in fact not at all what we want.

Here's a simple app, that returns immediatly, generically, leaving a very small footprint in memory while the called shell runs.

I confess, I haven't run valgrind on it and that might not be the only problem with it, though I have used it for years and will continue to use it unless or until something better turns up.

I called it "launch". And you probably won't like it. But I love it. :)

Here's the main function.

```
int main(int argc, char** argv)
{
  int err = 0;

  // help and version switches
  if(argc == 1)
    return usage(0);

  if(argc == 2)
  {
    if(strcmp(argv[1], "--help") == 0)
      return usage(0);
    if(strcmp(argv[1], "-v") == 0)
      return print_version(0);
  }

  err = parse_options(argc, argv);
  if(err)
  {
    printf("Unknown launch option '%s'\n", argv[err]);
  }
```

```
  // reassemble commandline
  char args[1024];
  err = unparse_args(args, 1000, argc, argv);
  if(err)
  {
    if(err == ENOMEM)
      fprintf(stderr, "Out of memory or args list too long\n");
    else if (err == E2BIG)
      fprintf(stderr, "Argument list too long\n");
    return 1;
  }

  // if -q suppress all feedback from the app
  if(!flg.warn)
  {
    fclose(stdout);
    fclose(stderr);
    strcat(args, " > /dev/null 2>&1");
  }

  // do the fork and execute the commandline
  pid_t pid;

  pid = fork ();
  if (pid == 0)
  {


    // command path, command name, arg1, arg2, ...
    execl ("/bin/sh", "sh", "-c", args, NULL);

    // _exit(code) for threads and forks.
    _exit (EXIT_FAILURE);
  }
  else if (pid < 0)
    return -1;
  else
    return 0;
}
```

And I'll attach the missing pieces as a tarball in case anyone is curious about this thing or has any ideas as to how to improve it.

The binary and sources, and makefile are all included. I believe the executable is the debug version.

If you watch the processes with gnome-system-monitor or ksysguard (or ps -ax or ???) you will see that the application runs under init and the footprint is very reasonable. The switches allow feedback but the default is to run silently.

Question: Is it possible to launch an invisible window to run these "shells" in UPP so that we get concurrently running applications without having to use external tools such as kshell<N> or 'launch" while retaining the ability to shut them down and possibly kill applications that hang if they exceed a reasonable timeout?

File Attachments

1) launch-1.4.tar.gz, downloaded 301 times