Send messages to program through command line

Asked 16 years, 4 months ago Modified 14 years, 4 months ago Viewed 4k times

9

I have this program, we'll call it Host. Host does all kinds of good stuff, but it needs to be able to accept input through the command line **while it's running**. This means it has to somehow send its other process data and then quit. For example, I need to be able to do this:

```
./Host --blahblah 3 6 3 5
```

This should somehow end up calling some function in Host called

```
handleBlahBlah(int x1, int y1, int x2, int y2){
  //do some more sweet stuff
}
```

Host is a C program, and does not need to support multiple instances.

An example of this is Amarok music player. With Amarok running and playing, you can type "amarok --pause" and it will pause the music.

I need to be able to do this in Linux or Windows. Preferably Linux.

What is the cleanest way to implement this?

c architecture command-line io

Share

Improve this question

Follow

edited Aug 11, 2010 at 16:58

sth

229k ● 56 ● 286 ● 368

asked Aug 13, 2008 at 22:40

andrewrk

31.1k ● 28 ● 95 ● 117

Are you wanting to pass it arguments when the program is already running? What OS is this being programmed for? – Adam Haile Aug 13, 2008 at 22:47

6 Answers

Sorted by: Highest score (default)

If you were on Windows, I'd tell you to use a hidden window to receive the messages, but since you used `./`, I assume you want something Unix-based.

10

In that case, I'd go with a named pipe. Sun has a tutorial about named pipes that might be useful.

The program would probably create the pipe and listen. You could have a separate command-line script which would open the pipe and just echo its command-line arguments to it.

You *could* modify your program to support the command-line sending instead of using a separate script. You'd do the same basic thing in that case. Your program would look at it's command-line arguments, and if applicable, open the pipe to the "main" instance of the program, and send the arguments through.

Share
Improve this answer
Follow

edited Aug 13, 2008 at 22:53

answered Aug 13, 2008 at 22:49

Derek Park
46.8k ● 16 ● 59 ● 76

1 +1 I've used this named pipe methodology several times and it works quite well. I've also used UNIX sockets for cases where the main program needed to return data to the command-line script/program. – jschmier Jun 7, 2010 at 15:45

6

If it needs to be cross-platform, you might want to consider making the running instance listen on a TCP port, and have the instance you fire up from the command-line send a message to that port.

Share Improve this answer Follow

answered Aug 13, 2008 at 22:51

Stu
15.8k ● 4 ● 45 ● 74

4

I suggest using either a Unix socket or D-Bus. Using a socket might be faster if you're familiar with Unix sockets programming and only want a few operations, whereas D-Bus might make it easier to concentrate on implementing the functionality in a familiar object-oriented way.

Take a look at Beej's Guide to Unix IPC, particularly the chapter on Unix sockets.

Share Improve this answer Follow

answered Aug 18, 2008 at 8:55

T Percival
8,654 ● 4 ● 44 ● 43

What no one has said here is this: "you can't get there from here".

1

The command line is only available as it was when your program was invoked.

The example of invoking "**fillinthename** arguments ..." to communicate with **fillinthename** once **fillinthename** is running can only be accomplished by using two instances of the program which communicate with each other.

The other answers suggest ways to achieve the communication.

An **amarok** like program needs to detect the existence of another instance of itself in order to know which role it must play, the major role of persistent message receiver/server, or the minor role of one shot message sender.

edited to make the word fillinthename actually be displayed.

Share
Improve this answer
Follow

edited Jun 7, 2010 at 14:55

answered Jun 7, 2010 at 14:47

pbernatchez
241 ● 1 ● 5

0

One technique I have seen is to have your `Host` program be merely a "shell" for your real program. For example when you launch your application normally (e.g.: `./Host`), the program will fork into the "main app" part of your code. When you launch your program in a way that suggests you want to signal the running instance (e.g.: `./Host --send-message restart`), the program will fork into the "message sender" part of your code. It's like having two apps in one. Another option that doesn't use `fork` is to make `Host` purely a "message sender" app and have your "main app" as a separate executable (e.g.: `Host_core`) that `Host` can launch separately.

The "main app" part of your program will need to open up some kind of a communication channel to receive messages, and the "message sender" part will need to connect to that channel and use it to send messages. There are several different options available for sending messages between processes. Some of the more common methods are pipes and sockets. Depending on your OS, you may have additional options available; for instance, QNX has channels and BeOS/Haiku have BMessages. You may also be able to find a library that neatly wraps up this functionality, such as lcm.

Share Improve this answer Follow

answered Aug 11, 2010 at 17:32

bta
45k ● 7 ● 72 ● 100

So, I may be missing the point here, but by deafult a C program's main function takes two arguments; argc, a count of the number of arguments (at least one), and argv (or

-2

arg vector), the argument list. You could just parse through the arguments and call the correct method. For example:

```
int main(int argc, *argv[])
{
    /*loop through each argument and take action*/
     while (--argc > 0)
     {
          printf(%s%s, *++argv, (argc > 1) ? " " : "");
     }
}
```

would print all of the arguments to screen. I am no C guru, so I hope I haven't made any mistakes.

EDIT: Ok, he was after something else, but it wasn't really clear before the question was edited. Don't have to jump on my rep...

Share

Improve this answer

Follow

edited Aug 13, 2008 at 22:52

answered Aug 13, 2008 at 22:43

Ed Swangren

125k • 24 • 188 • 268