

Re: Threads Vs Forks in Embedded Environment

Source: <http://unix.derkeiler.com/Newsgroups/comp.unix.programmer/2006-09/msg00116.html>

- *From:* "abhisheksingh.bits@xxxxxxxx" <abhisheksingh.bits@xxxxxxxx>
 - *Date:* 6 Sep 2006 01:40:02 -0700
-

i want to have a specific differences between threads and forks in embedded environment to convince me that i can use threads or forks....because using thread or forks are 2 different design alternatives of my project. I have found out some differences but for normal computing environment. these are as follows:

The advantage of threads, generic and project specific:

The advantage of threads is their lower resource consumption.

It takes much less CPU time to switch among threads than between processes, because there's no need to switch address spaces. In addition, because they share address space, threads in a process can communicate more easily with one another.

The Disadvantage of threads:

Project Specific:

Threads require support libraries, so extra space is required in flash memory.

Updation of libraries may also be required so this may increase the installation time.

Though threads share resources, in our case the sharing is not substantial.

Generic:

Programs using threads are harder to write and debug. Not all programming libraries are designed for use with threads. And not all legacy applications work well with threaded applications. Some programming tools also make it harder to design and test threaded code.

Re: Threads Vs Forks in Embedded Environment

Another problem is concurrency and synchronization complexity. sharing, locking, deadlock, race conditions come vividly alive in threads. Processes don't usually have to deal with this, since most shared data is passed through pipes. Threads can share file handles, variables, signals, etc. this may lead to error conditions if not handled properly.

Applications executed in a thread environment must be thread-safe. This means that functions (or the methods in object-oriented applications) must be reentrant—a function with the same input always returns the same result, even if other threads concurrently execute the same function. Accordingly, functions must be programmed in such a way that they can be executed simultaneously by several threads.

Thread-related bugs can also be more difficult to find. Threads in a process can interfere with one another's data. The operating system may limit how many threads can perform operations, such as reading and writing data, at the same time. Scheduling different threads to avoid conflicts can be a difficult.

Systems like windows and OS/2 are said to have "cheap" threads and "expensive" processes, while in other OS there is not a big difference.

.