exectimes

cooperative limiting of concurrent instances of a specified program

Usage

exectimes *lock-file max-instances command command-args ...*
exectimes *lock-file* check
exectimes *lock-file* list

Synopsis

The `exectimes` program uses POSIX advisory locking to prevent more than a specified number of instances of a program from running at one time.

A typical invocation looks something like this:

```
exectimes /var/run/mylockfile 10 command-to-run command-arguments
```

which is usually placed in a shell script used to launch the program.

Details

The first argument to `exectimes` is the lock file to use which will store the number of in-use locks corresponding to the number of running instances of the target program. The second argument is the maximum number of instances of the target program allowed to run. The target program and its arguments follow.

If there are less than the maximum number of instances of the target program running, the target program will be started. Otherwise a message indicating the program cannot start due to the number of concurrent instances is printed.

If the second argument is "check", the lock file will just be examined to determine and report the number of currently running instances.

If the second argument is "list", a list of the currently running instances is displayed which includes the slot number (corresponding to the position of the associated lock byte, see the Implementation section below) and the PID of each process.

Note that there is nothing tying the lock file to a specific command, a separate lock file should be used with each different command regulated by `exectimes` unless a number of different commands should share a pool of available instances.

Implementation

`exectimes` will execute the program only if the number of currently running instances, determined by examining the specified the lock file, is less than the maximum allowed as specified as the second program argument.

The lockfile is created by the program if it does not exist and the program then attempts to lock the initial bytes of the file. Once it has this lock, it reads the data in this section which tells it the index of the highest-numbered lock ever taken out so far. It then checks each byte up to this max index and determines how many of those bytes are locked. Each locked byte indicates a process that is running that should be counted towards the max allowed. If the number of locked bytes is less than the number of allowed concurrent instances, the first available unlocked byte is locked, the max index is updated if necessary, and the lock on the initial section is released. The program then execs the new process.

Since locks are carried over to new processes through exec, the lock will stay in place until the program terminates. Since the operating system automatically removes the lock when the process terminates, no modification needs to be made to the target process and the lock will be released even if the process is killed or terminated abnormally. Since the program doesn't do anything without having a lock on the initial part of the file, there are no race conditions. Lastly, since POSIX allows the locking of bytes that do not exist, the only actual data in the lock file is the index of the maximum used byte.