Coding Challenge - Divyendra Patil

Resume: https://divyendra.com/resume/DivyendraPatil_Resume.pdf

Although the program is commented, this is a readme for the process I went through. I have no written C programs outside school coursework and since on the call, it was hinted that it is harder to write the daemon in C, I thought of giving it a try as it will be a learning experience in itself.

Ended up reading a bit on what it takes for a program to become a daemon. Was not aware of what daemonizing library were either so had to look them up.

For the current program written, I decided to log everything inside syslog itself but can be written in some other file if need. Getting it to run only one instance was tricky. I was executing the code on a Mac and later came to an understanding that it might not work, created a ubuntu docker container for the same but it again had its own limitations on systemctl and no log files.

So finally decided to create a ubuntu VM and test/run everything.

Even if we do not go ahead with the application, I would like to have feedback on the program as it was a good refresher and a learning experience.

==========

All of the program functions are at the beginning.
The first one is a `sig_handler` which writes the status of which signal was sent to the daemon will be written to syslog and executed.

Then there is `write_status` which writes different statuses to syslog, mainly if the pid is less than or greater than “0”, if the “/“ or “/usr” directory exists or no and if the Sid is less than “0”.

In `main`() , we first start with the signal handling (Although deprecated and we can use `sigaction` which has numerous advantages over `signal`, I just wanted to keep it simple for this program)

We define the signal's to be caught (Can be defined more). The signal handler function has void return type and accepts a signal number corresponding to the signal that needs to be handled. And here we have user defined signals that can be sent and handled.

There are different ways of ensuring that only one instance is run (mutex, using unix domain socket etc), for the simplicity of it, I decided to go with the locking mechanism to ignore stale pid files (we don't have to delete them). When the program will terminate for any reason the OS releases the file lock for us. Pid files are not terribly useful because they can be stale (the file exists but the process does not). Hence, the program executable itself can be locked instead of creating and locking a pid file.

So the next block of code opens and creates a lock file, and using flock, gets a `return_code` which determines if the lock is present or no.

Then, wherever we want to declare a variable that is going to be deal with the process ids we can use pid_t data type.
We then fork off the parent process and save a copy and terminate from the parent process. Later, create a new SID for the child process and save a copy. We then fork again for PID to not be same as SID. The reason is that our process can't take control of a TTY again due to the second fork.

The explanation for umask is commented in the program itself.

Then there is an infinite loop to check "/usr" directory and write to syslog if it exists.

How to run:

```
1] Compile the code: clang daemon_program.c -o
daemon_program
2] Start the daemon:./daemon_program
3] Check if everything is working properly: ps -xj |
grep daemon_program
4] Again run the program.
5] Only one instance of the program will still be
running.

The parent process ID (PPID) is 1 (The init process)
And PID != SID (Explained the reasoning above)
```

```
root@divyendra-virtual-machine: ~
File  Edit  View  Search  Terminal  Tabs  Help
root@divyendra-virtual-machine: ~        root@divyendra-virtual-machine: ~
root@divyendra-virtual-machine:~# clang daemon_program.c -o daemon_program
daemon_program.c:178:5: warning: implicit declaration of function 'umask' is invalid in C99 [-Wimplicit-function-declaration]
    umask(0);
    ^
1 warning generated.
root@divyendra-virtual-machine:~# ./daemon_program
root@divyendra-virtual-machine:~# ps -xj | grep daemon_program
   934   2078   2077   2077 ?           -1 S        0   0:00 ./daemon_program
  1770   2080   2079   1709 pts/0      2079 S+       0   0:00 grep --color=auto daemon_program
root@divyendra-virtual-machine:~# ./daemon_program
root@divyendra-virtual-machine:~# kill -USR1 2078
root@divyendra-virtual-machine:~#
```

While the syslog:

```
daemon_program[2076]: Process ID is greater than 0.. starting to log file
daemon_program[2077]: Process ID is greater than 0.. starting to log file
daemon_program[2078]: Directory exists
daemon_program: Daemon already running. Stopping this instance and letting the first one to run
daemon_program[2078]: SIGUSR Received. Stopping daemon
```

These are few resources I read and implemented the program.

http://www.linfo.org/daemon.html
http://www.netzmafia.de/skripten/unix/linux-daemon-howto.html
https://www.gnu.org/software/libc/manual/html_node/Submitting-Syslog-Messages.html#Submitting-Syslog-Messages
http://fibrevillage.com/sysadmin/70-how-to-syslog-your-program-output
https://www.tutorialspoint.com/c_standard_library/c_function_signal.htm