Log Book Week 11

Jose Devian Hibono
1706039603
System Programming - C

This week’s material is about daemon & background processes. First I did the pre-test using the resources on the internet. Here are the resources I gathered, I tried to write my own understandings in this log book.

Daemon Process

A daemon process is a background process that is not under the user's direct control. Usually, this process starts when the system is bootstrapped and the system is shut down. The init mechanism is usually the parent method of the daemon process. This is because after the parent process forks the daemon process and terminates, the init system typically adopts the daemon process.

Examples

- cron
- sshd
- httpd
- inetd – listens for incoming connections
- hostapd

Background Process

A process that works behind the scenes (i.e. in the background) and without user interaction is a background process. Typical tasks include logging, device management, scheduling, and user notification for these processes.

A background process like this is still parented, s o signals to that process group will continue to it. If the parent process terminates, the children will receive signals that will most likely terminate them, as well. While in a daemon process, since there’s no parent process, the daemon will stay put.

Daemonize

1. Call umask to set the development mask of the file mode to a known value, normally 0.

2. Fork the call to have a parent exit.

3. To create a new session, please call setsid.
   The process:
   - becomes the leader of a new session,
   - becomes the leader of a new process group,
   - and is disassociated from its controlling terminal.

4. Modify the existing working directory to the root folder.

5. A installed file system could be the current working directory inherited from the parent. Since daemons usually exist until the system is rebooted, it is difficult to unmount the file system if the daemon remains on the installed file system.

6. Descriptors of unneeded files should be locked. This prevents any descriptors that it might have inherited from its parent from keeping open the daemon (which could be a shell or some other process).
7. Some daemons open the 0, 1, and 2 file descriptors to /dev/null so that there will be no effect on any library routines attempting to read from standard input or write to standard output or standard error.

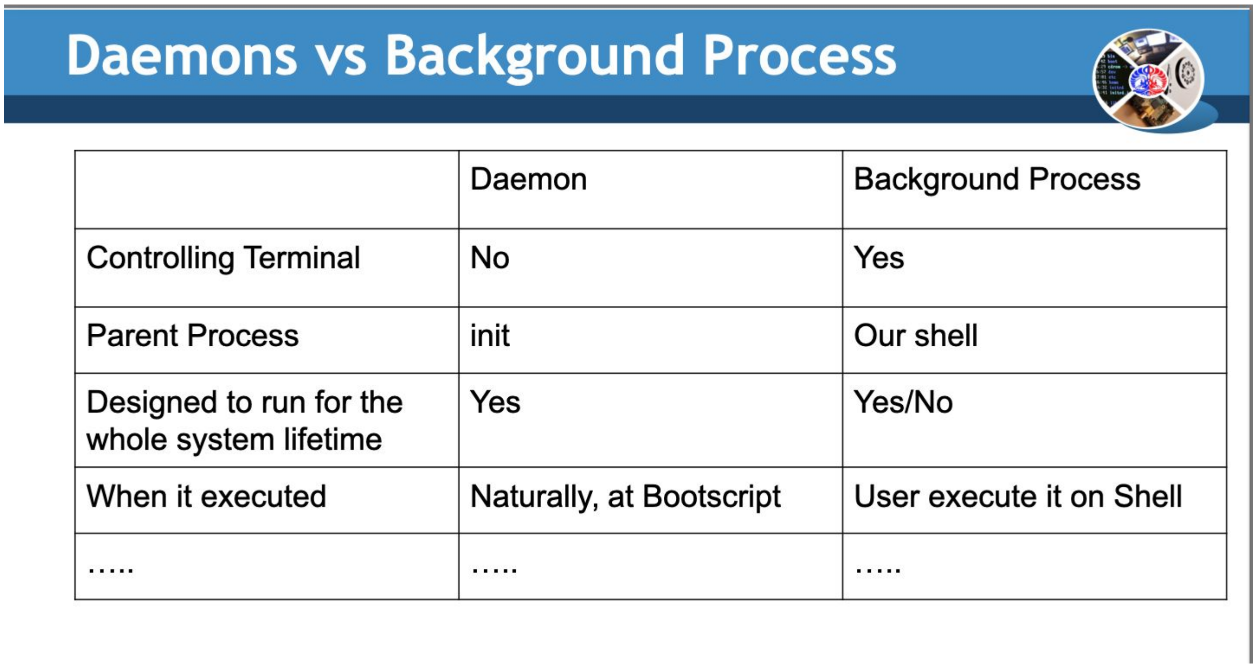
Error Logging

One problem a daemon has is how to handle error messages. It cannot (simply) write to:
- Standard error: the controlling terminal should not be on it.
- Console device: The console device runs a window system on several workstations.
- Separate files: keeping up which daemon writes to which log file and reviewing these files on a regular basis is a hassle.

There are three ways to generate log messages:
1. The log function can be named by kernel routines. Any user process which opens and reads the device /dev/klog can read these messages.
2. To produce log messages, most user processes (daemons) call the syslog(3) feature. This allows the message to be sent to the datagram socket /dev/log for the UNIX domain.
3. Log messages may be sent to UDP port 514 by a user process on this host or any other host connected to this host by a TCP/IP network. Note that the syslog function never

generates these UDP datagrams: they require explicit network programming by the process generating the log message.

All three types of log messages are read in the syslogd daemon. This daemon reads a configuration file at start-up, usually /etc/syslog.conf, which decides where to send various groups of messages. For example, you can send (if logged in) urgent messages to the system administrator and print them on the console, while alerts can be logged to a file.



Daemons vs Background Process

	Daemon	Background Process
Controlling Terminal	No	Yes
Parent Process	init	Our shell
Designed to run for the whole system lifetime	Yes	Yes/No
When it executed	Naturally, at Bootscript	User execute it on Shell
.....	.....	.....

Taken from the course's slides.

Application in the real world

Client-server Model

A common usage is as a server process for a daemon process. The Syslogd process is a server that uses a UNIX domain datagram socket to transmit messages to it through user processes (clients).

A server is a process that waits to be contacted by a client, asking for some sort of service. Logging of an error message is the service offered by syslogd server.

In order to provide service to a client, servers typically fork and execute another program. Multiple file descriptors also handle these servers (e.g. communication endpoints, configuration files, log files). In the child method, it would be irresponsible to leave these file descriptors open because they will probably not be used in the child executed program especially if the program is unrelated to the server. At worst, leaving them open could pose a security issue: the executed program could do something malicious, such as altering the configuration file of the server or tricking the client into believing that it is interacting with the server, gaining access to unauthorized data.

Setting the close-on-exec flag for all file descriptors that the executed program won't need is a simple solution to this problem.