Service and Process Management

In general, there are two types of services: internal, the relevant services that are required at system startup, which for example, perform hardware-related tasks, and services that are installed by the user, which usually include all server services. Such services run in the background without any user interaction. These are also called `daemons` and are identified by the letter '`d`' at the end of the program name, for example, `sshd` or `systemd`.

Most Linux distributions have now switched to `systemd`. This daemon is an `Init process` started first and thus has the process ID (PID) 1. This daemon monitors and takes care of the orderly starting and stopping of other services. All processes have an assigned PID that can be viewed under `/proc/` with the corresponding number. Such a process can have a parent process ID (PPID), and if so, it is known as the child process.

Besides `systemctl` we can also use `update-rc.d` to manage SysV init script links. Let us have a look at some examples. We will use the `OpenSSH` server in these examples. If we do not have this installed, please install it before proceeding to this section.

Systemctl

After installing `OpenSSH` on our VM, we can start the service with the following command.

```
amit8986@htb[/htb]$ systemctl start ssh
```

After we have started the service, we can now check if it runs without errors.

```
amit8986@htb[/htb]$ systemctl status ssh

● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2020-05-14 15:08:23 CEST; 24h ago
   Main PID: 846 (sshd)
   Tasks: 1 (limit: 4681)
   CGroup: /system.slice/ssh.service
           └─846 /usr/sbin/sshd -D

Mai 14 15:08:22 inlane systemd[1]: Starting OpenBSD Secure Shell server...
Mai 14 15:08:23 inlane sshd[846]: Server listening on 0.0.0.0 port 22.
Mai 14 15:08:23 inlane sshd[846]: Server listening on :: port 22.
Mai 14 15:08:23 inlane systemd[1]: Started OpenBSD Secure Shell server.
Mai 14 15:08:30 inlane systemd[1]: Reloading OpenBSD Secure Shell server.
Mai 14 15:08:31 inlane sshd[846]: Received SIGHUP; restarting.
Mai 14 15:08:31 inlane sshd[846]: Server listening on 0.0.0.0 port 22.
Mai 14 15:08:31 inlane sshd[846]: Server listening on :: port 22.
```

To add OpenSSH to the SysV script to tell the system to run this service after startup, we can link it with the following command:

```
amit8986@htb[/htb]$ systemctl enable ssh

Synchronizing state of ssh.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable ssh
```

Once we reboot the system, the OpenSSH server will automatically run. We can check this with a tool called `ps`.

```
amit8986@htb[/htb]$ ps -aux | grep ssh

root       846  0.0  0.1  72300  5660 ?        Ss   Mai14   0:00 /usr/sbin/sshd -D
```

We can also use `systemctl` to list all services.

```
amit8986@htb[/htb]$ systemctl list-units --type=service

UNIT                                                      LOAD   ACTIVE SUB     DESCRIPTION
accounts-daemon.service                                   loaded active running Accounts Service
acpid.service                                             loaded active running ACPI event daemon
apache2.service                                           loaded active running The Apache HTTP Server
apparmor.service                                          loaded active exited  AppArmor initialization
apport.service                                            loaded active exited  LSB: automatic crash repor
avahi-daemon.service                                      loaded active running Avahi mDNS/DNS-SD Stack
bolt.service                                              loaded active running Thunderbolt system service
```

It is quite possible that the services do not start due to an error. To see the problem, we can use the tool `journalctl` to view the logs.

```
amit8986@htb[/htb]$ journalctl -u ssh.service --no-pager

-- Logs begin at Wed 2020-05-13 17:30:52 CEST, end at Fri 2020-05-15 16:00:14 CEST. --
Mai 13 20:38:44 inlane systemd[1]: Starting OpenBSD Secure Shell server...
Mai 13 20:38:44 inlane sshd[2722]: Server listening on 0.0.0.0 port 22.
Mai 13 20:38:44 inlane sshd[2722]: Server listening on :: port 22.
Mai 13 20:38:44 inlane systemd[1]: Started OpenBSD Secure Shell server.
Mai 13 20:39:06 inlane sshd[3939]: Connection closed by 10.22.2.1 port 36444 [preauth]
Mai 13 20:39:27 inlane sshd[3942]: Accepted password for master from 10.22.2.1 port 36452 ssh2
Mai 13 20:39:27 inlane sshd[3942]: pam_unix(sshd:session): session opened for user master by (uid=0)
Mai 13 20:39:28 inlane sshd[3942]: pam_unix(sshd:session): session closed for user master
Mai 14 02:04:49 inlane sshd[2722]: Received signal 15; terminating.
Mai 14 02:04:49 inlane systemd[1]: Stopping OpenBSD Secure Shell server...
Mai 14 02:04:49 inlane systemd[1]: Stopped OpenBSD Secure Shell server.
-- Reboot --
```

Kill a Process

A process can be in the following states:

- `Running`
- `Waiting (waiting for an event or system resource)`
- `Stopped`
- `Zombie (stopped but still has an entry in the process table).`

Processes can be controlled using `kill`, `pkill`, `pgrep`, and `killall`. To interact with a process, we must send a signal to it. We can view all signals with the following command:

```
amit8986@htb[/htb]$ kill -l

 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

The most commonly used are:

Signal	Description
1	`SIGHUP` - This is sent to a process when the terminal that controls it is closed.
2	`SIGINT` - Sent when a user presses `[Ctrl] + C` in the controlling terminal to interrupt a process.
3	`SIGQUIT` - Sent when a user presses `[Ctrl] + D` to quit.
9	`SIGKILL` - Immediately kill a process with no clean-up operations.
15	`SIGTERM` - Program termination.
19	`SIGSTOP` - Stop the program. It cannot be handled anymore.
20	`SIGTSTP` - Sent when a user presses `[Ctrl] + Z` to request for a service to suspend. The user can handle it afterward.

For example, if a program were to freeze, we could force to kill it with the following command:

```
amit8986@htb[/htb]$ kill 9 <PID>
```

Background a Process

Sometimes it will be necessary to put the scan or process we just started in the background to continue using the current session to interact with the system or start other processes. As we have already seen, we can do this with the shortcut `[Ctrl + Z]`. As mentioned above, we send the `SIGTSTP` signal to the kernel, which suspends the process.

```
amit8986@htb[/htb]$ ping -c 10 www.hackthebox.eu

amit8986@htb[/htb]$ vim tmpfile
[Ctrl + Z]
[2]+  Stopped                 vim tmpfile
```

Now all background processes can be displayed with the following command.

```
amit8986@htb[/htb]$ jobs

[1]+  Stopped                 ping -c 10 www.hackthebox.eu
[2]+  Stopped                 vim tmpfile
```

The `[Ctrl] + Z` shortcut suspends the processes, and they will not be executed further. To keep it running in the background, we have to enter the command `bg` to put the process in the background.

```
amit8986@htb[/htb]$ bg

amit8986@htb[/htb]$
--- www.hackthebox.eu ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 113482ms

[ENTER]
[1]+  Exit 1                  ping -c 10 www.hackthebox.eu
```

Another option is to automatically set the process with an AND sign (`&`) at the end of the command.

```
amit8986@htb[/htb]$ ping -c 10 www.hackthebox.eu &

[1] 10825
PING www.hackthebox.eu (172.67.1.1) 56(84) bytes of data.
```

Once the process finishes, we will see the results.

```
amit8986@htb[/htb]$

--- www.hackthebox.eu ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9210ms

[ENTER]
[1]+  Exit 1                  ping -c 10 www.hackthebox.eu
```

Foreground a Process

After that, we can use the `jobs` command to list all background processes. Backgrounded processes do not require user interaction, and we can use the same shell session without waiting until the process finishes first. Once the scan or process finishes its work, we will get notified by the terminal that the process is finished.

```
amit8986@htb[/htb]$ jobs

[1]+  Running                 ping -c 10 www.hackthebox.eu &
```

If we want to get the background process into the foreground and interact with it again, we can use the `fg <ID>` command.

```
amit8986@htb[/htb]$ fg 1
ping -c 10 www.hackthebox.eu

--- www.hackthebox.eu ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9206ms
```

Execute Multiple Commands

There are three possibilities to run several commands, one after the other. These are separated by:

- `Semicolon (;)`
- `Double ampersand characters (&&)`
- `Pipes (|)`

The difference between them lies in the previous processes' treatment and depends on whether the previous process was completed successfully or with errors. The semicolon (`;`) is a command separator and executes the commands by ignoring previous commands' results and errors.

```
amit8986@htb[/htb]$ echo '1'; echo '2'; echo '3'

1
2
3
```

For example, if we execute the same command but replace it in second place, the command `ls` with a file that does not exist, we get an error, and the third command will be executed nevertheless.

```
amit8986@htb[/htb]$ echo '1'; ls MISSING_FILE; echo '3'

1
ls: cannot access 'MISSING_FILE': No such file or directory
3
```

However, it looks different if we use the double AND characters (`&&`) to run the commands one after the other. If there is an error in one of the commands, the following ones will not be executed anymore, and the whole process will be stopped.

```
amit8986@htb[/htb]$ echo '1' && ls MISSING_FILE && echo '3'

1
ls: cannot access 'MISSING_FILE': No such file or directory
```

Pipes (`|`) depend not only on the correct and error-free operation of the previous processes but also on the previous processes' results.

eu-academy-1

PROTOCOL

[✓ UDP 1337] TCP 443

DOWNLOAD VPN CONNECTION FILE

Start Instance

1 / 1 spawns left

Waiting to start...

Questions

Answer the question(s) below to complete this Section and earn cubes!

Target: 10.129.169.240

Life Left: 119 minutes

Cheat Sheet

Download VPN Connection File

SSH to 10.129.169.240 with user "htb-student" and password "HTB_@cademy_stdnt!"

+ 1 Use the "systemctl" command to list all units of services and submit the unit name with the description "Load AppArmor profiles managed internally by snapd" as the answer.

snapd.apparmor.service

Submit Hint

Previous Next

Mark Complete & Next

Cheat Sheet

Go to Questions