4. Print the machine type as follows:

```
$ uname -m
```

5. In order to print details about the CPU, use:

```
$ cat /proc/cpuinfo
```

   In order to extract the processor name, use:

```
$ cat /proc/cpuinfo | sed -n 5p
```

   The fifth line contains the processor name.

6. Print details about the memory or RAM as follows:

```
$ cat /proc/meminfo
```

   Print the total memory (RAM) available on the system as follows:

```
$ cat /proc/meminfo  | head -1
MemTotal:        1026096 kB
```

7. In order to list out the partitions information available on the system, use:

```
$ cat /proc/partitions
```

   Or:

```
$ fdisk -l  #If you don't get any output, run as root
```

8. Get the entire details about the system as follows:

```
$ lshw   #Recommended to run as root
```

Using /proc for gathering information

`/proc` is an in-memory pseudo filesystem available with the GNU/Linux operating system. It was actually introduced to provide an interface to read several system parameters from the user space. It is very interesting and we can gather lots of information from it. Let's see how to.

How to do it...

If you look at `/proc`, you will see several files and directories, some of which are already explained in other recipes in this chapter. You can simply `cat` files in `/proc` and the subdirectories to get information. All of them are well-formatted text.

There will be a directory in `/proc` for every process that is running on the system, named after the PID of that process.

Suppose Bash is running with PID 4295 (`pgrep bash`), `/proc/4295` will exist. Each of the directories corresponding to the process will contain a lot of information regarding to that process. Few of the important files in `/proc/PID` are as follows.

- `environ`: This contains environment variables associated with that process. `cat /proc/4295/environ` will display all the environment variables passed to that process.
- `cwd`: This is a symlink to a working directory of the process.
- `exe`: This is a symlink to the running executable for the current process.

  ```
  $ readlink /proc/4295/exe
  /bin/bash
  ```

- `fd`: This is the directory consisting of entries on file descriptors used by the process.

Scheduling with cron

It is a common requirement to schedule execution of scripts at a given time or at given time intervals. The GNU/Linux system comes with different utilities for scheduling tasks. `cron` is such a utility that allows tasks to automatically run in the background of the system at regular intervals using the `cron` daemon. The `cron` utility makes use of a file called **cron table** that stores a list of schedules of scripts or commands to be executed and the time at which they are to be executed. A common example usage is that you can schedule downloads of files from the Internet during the free hours (certain ISPs provide free usage hours, usually, at night time). This way you won't be required to wake up in the night to start the download. In addition to writing a `cron` entry and schedule the download, you can also schedule to drop the Internet connection automatically and shutdown the system when the free usage hours end.

Getting ready

The `cron` scheduling utility comes with all the GNU/Linux distributions by default. Once we write the cron table entry, the commands will be executed at the time specified for execution. The command `crontab` is used to add jobs to the cron table. The cron table is a simple text file and each user has a separate copy.

How to do it...

In order to schedule tasks, we should know the format for writing the cron table. A cron job specifies the path of a script or command to be executed and the time at which it is to be executed.

1. cron job to execute the `test.sh` script at the second minute of all hours on all days:

   ```
   02 * * * * /home/slynux/test.sh
   ```