```
^C
bob@alice:$ df -k /var
Filesystem        1K-blocks   Used Available Use% Mounted on
/dev/sda1           9833300   2648816   6684980  29% /

bob@alice:$ quota
Disk quotas for user bob (uid 1001):
Filesystem blocks  quota  limit grace files quota limit grace
/dev/sda1  99996   100000 100000      128   0     0
```

Problem 4.22 Which resources are exhausted in this attack? Why is the user's quota not debited when the links are created? What can the adversary accomplish with such an attack? How can this problem be solved?

Whenever possible, world writable directories should be moved to a partition separate from the operating system. Otherwise, depending on the system's implementation, an adversary who is able to fill the entire file system as described above could successfully mount a denial-of-service attack. For a system-wide temporary directory, a RAM disk could be used.

4.6 Change Root

A further possibility to restrict permissions is *Change Root* (chroot), in which a process is "jailed" in a subdirectory of the file system.

▷ Read the manual page for `chroot`.

In order to jail a process in a new root directory, we will have to provide all the necessary commands we want the process to be able to execute within this chroot environment. As an example, we will execute the simple script `chroot-test.sh`, which only returns a listing of the root directory of the system.

Note that in this section the term root does not refer to the superuser *`root`* but to the root node of the file system.

▷ Create a simple script `/home/alice/chroot-test.sh`, which contains only the command `ls /`. Then make it executable and execute it.

```
alice@alice:$ /home/alice/chroot-test.sh
```

Now, we will jail the script in the `/home` directory. Hence, we will have to provide the binaries and necessary libraries for `bash` and `ls` in the new environment. We therefore first create a `bin` and a `lib` directory and then copy all the necessary files. Note that by using the command `lld` a binary's dependencies can be examined.

▷ Build a minimal chroot environment by performing the following commands.

```
alice@alice:$ sudo mkdir /home/bin /home/lib
alice@alice:$ sudo cp /bin/bash /bin/ls /home/bin

alice@alice:$ ldd /bin/bash
linux-gate.so.1 =>  (0x00695000)
libncurses.so.5 => /lib/libncurses.so.5 (0x00947000)
libdl.so.2 => /lib/tls/i686/cmov/libdl.so.2 (0x0082d000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x00a8c000)
/lib/ld-linux.so.2 (0x00535000)

alice@alice:$ sudo cp /lib/libncurses.so.5 /lib/libdl.so.2 \
> /lib/libc.so.6 /lib/ld-linux.so.2 /home/lib

alice@alice:$ ldd /bin/ls
linux-gate.so.1 =>  (0x0055c000)
librt.so.1 => /lib/tls/i686/cmov/librt.so.1 (0x00901000)
libselinux.so.1 => /lib/libselinux.so.1 (0x00715000)
libacl.so.1 => /lib/libacl.so.1 (0x00819000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x009a2000)
libpthread.so.0 => /lib/tls/i686/cmov/libpthread.so.0
(0x007f8000)
/lib/ld-linux.so.2 (0x00f93000)
libdl.so.2 => /lib/tls/i686/cmov/libdl.so.2 (0x00115000)
libattr.so.1 => /lib/libattr.so.1 (0x00989000)

alice@alice:$ sudo cp /lib/librt.so.1 /lib/libselinux.so.1 \
> /lib/libacl.so.1 /lib/libpthread.so.0 /lib/libattr.so.1 \
> /home/lib
```

You can similarly define additional commands that you want the jailed process to be able to execute.

▷ Execute the `chroot-test.sh` script again, but this time in the jail. Be aware of the fact that the path to the file differs, since we set the root to `/home`.

```
alice@alice:$ sudo chroot /home /alice/chroot-test.sh
alice  bin  bob  lib
```

As you can see, the output reflects that the script was executed in the defined chroot environment. Since we also copied bash and its dependencies, you can also execute an interactive shell in the jail and try to escape.

```
alice@alice:$ sudo chroot /home bash
bash-4.1#
```

Problem 4.23 What are the security-relevant advantages of chroot environments, for example, for a server program?

We will now apply this mechanism to the remote access on **alice**. OpenSSH offers the opportunity to jail the users that are logging in remotely by restricting the files and directories that they may access.

▷ Read the manual page for `sshd_config`.

In what follows, we will restrict SSH access on **alice** in such a way that any authorized user is jailed in the directory `/home`. Thus, a user accessing **alice** remotely will not be able to leave the `/home` directory and is provided only with a minimal set of commands. We will use the change root environment built above.

▷ Append `ChrootDirectory /home` to the `sshd_config` file. Note that you need to be *root* to execute the next command.

```
root@alice:# echo "ChrootDirectory /home" >> \
> /etc/ssh/sshd_config
```

▷ Restart sshd now and log in to **alice** from another machine, for example, from **bob**.

```
alice@alice:$ sudo /etc/init.d/ssh restart

bob@bob:$ ssh alice@alice
alice@alice's password: *****
```

You are now logged in on **alice**, but restricted to `/home` as the root directory. You could use this mechanism to build more complicated environments. For example, in `/etc/ssh/sshd_config` you could chroot to `%h`, which refers to the current user's home directory. Then, analogously to the above, you could create different environments for every user and thus restrict them to specific sets of allowed commands within their own home directories.

Problem 4.24 What problem arises when building sophisticated chroot environments? What difficulties do you expect in terms of the administration and the operation of such environments?

Note that whenever an adversary gains superuser access within a chroot environment, there are different possibilities to escape from the jail. One is based on the fact that not all file descriptors are closed when calling `chroot()`. A simple C program